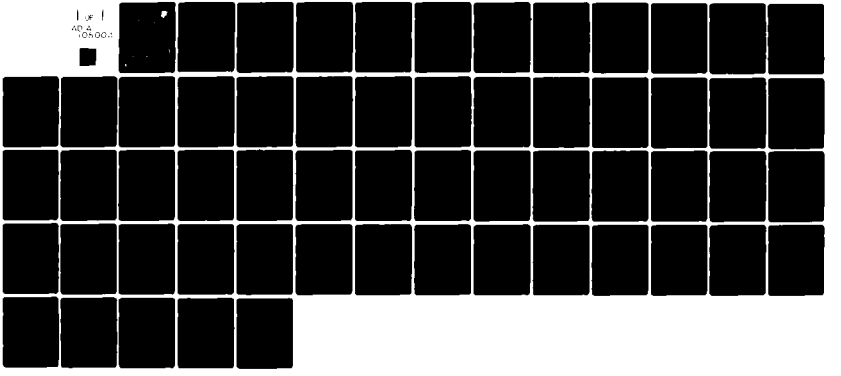
AD-A105 004 POLYTECHNIC INST OF NEW YORK BROOKLYN F/G 9/2
SOFTWARE MODELING STUDIES. VOLUME II. THE POLYNOMIAL MEASURE OF--ETC(U)
JUL 81 H RUSTON F30602-78-C-0057
UNCLASSIFIED RADC-TR-81-183-VOL-2 NL



RADC-TR-81-183, Vol II (of four)
Final Technical Report
July 1981

SOFTWARE MODELING STUDIES
THE POLYNOMIAL MEASURE OF COMPLEXITY

Polytechnic Institute of New York

Henry Ruston

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-81-183, Vol II (of three) has been reviewed and is approved for publication.

APPROVED: Rocco F. Iuorno

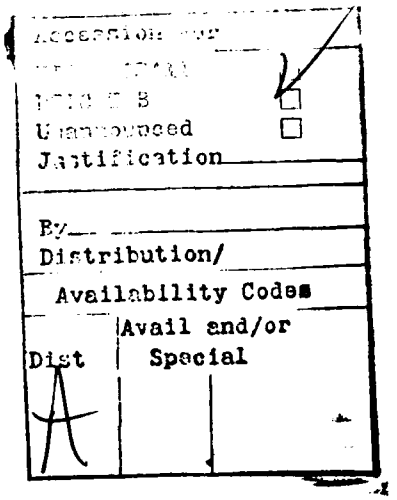
ROCCO F. IUORNO
Project Engineer

APPROVED:

JOHN J. MARCINIAK, Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office



If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIE) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-81-183, Vol II (of four)	2. GOVT ACCESSION NO. AD-A105 004	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE MODELING STUDIES V I THE POLYNOMIAL MEASURE OF COMPLEXITY		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report January 78 — October 80 6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) Henry Ruston		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0057
9. PERFORMING ORGANIZATION NAME AND ADDRESS Polytechnic Institute of New York 333 Jay Street Brooklyn NY 11201		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 23043401
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		12. REPORT DATE July 1981 13. NUMBER OF PAGES 60
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Rocco F. Iuorno (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Control Flow	Flowcharts
Cyclomatic Measure	Program Path
Program Structure	Cyclomatic Complexity
Polynomial Measure	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new measure of software complexity is introduced, one which describes a flow chart by a polynomial. This measure takes both the elements of a flow chart and its structure into account. Rules are given for obtaining the polynomials for various types of flow charts. The polynomial complexity measure can be used to compare alternate software designs and to divide a computer program into modules to obtain a minimal overall complexity. A comparison is made of this measure with several known complexity measures.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (cont'd.)

LIST OF FIGURES (cont'd.)

LIST OF FIGURES (cont'd.)

LIST OF FIGURES (cont'd.)

ABSTRACT

A new measure of complexity is introduced, one which describes a flowchart by a polynomial. This measure takes both the elements of the flow chart and its structure into account.

Rules are given for obtaining the polynomials for various flowcharts, such as structured, unstructured, with loops, and with simple or multiple selectors (i.e., CASE statements). The polynomial measure allows the comparison of alternate designs and tells how to divide a program into modules for minimal overall complexity. The measure also gives the number of path tests and bounds on cyclomatic complexity. Finally a comparison is made of this measure with several other popular complexity measures.

1.0 INTRODUCTION

A problem of interest in software engineering is the measuring of program *complexity*.

Complexity is a fuzzy and an ill defined concept, generally meaning that the program is "complicated" [1]. It is perceived that it is this "complication" which reduces a programmer's productivity from say 20 statements/day on job A to just 5 statements/day on job B. We perceive that it is the higher complexity of job B which accounts for this diminished productivity.

An accepted measure of complexity will allow to compare programs. One important application could be the measurement of complexities of alternate designs and the selection of the design with the lowest complexity.

There have been several attempts to quantify precisely the notion of complexity. Typically a countable property of a program is identified, and the count is defined as the complexity [1]. The control flow, in particular, has been a subject of extensive scrutiny. One example of such a measure is the cyclomatic number [2] counting the number of regions in a graph, and related to the number of deciders in the program.

The polynomial measure introduced here also arises from the properties of the control flow. It is quantitative, objective, and relatively simple to obtain. It tells about the structure of a program, and in fact allows the construction of a flowchart from the measure. It is related to the testing effort. Because it is the testing which consumes most labor, a measure related to testing reflects the total labor needed to produce the program.

2.0 THE STATE OF THE ART - A BRIEF REVIEW

Several measures of complexity have been introduced in the recent literature. We will briefly review several of the measures and their derivatives.

2.1 Complexity Based Upon Features of the Program [3].

Such measure counts the number of certain features of a program (for example, the number of IFs), and compares them with the number of same features in a "reference" program. The reference program is to be either given in an installation or obtained by averaging a group of programs. Different users may obtain different complexity measures by this technique, and as stated the measure only records the number of features and not their role in the program structure.

2.2 Complexity Based Upon Operator and Operand Counts

Halstead [4] derived several measures from the counts of distinct operators and operands in a program, and from the frequencies of each operator and each operand. He obtained an expression for the size of the implementing algorithm (called the program volume), and for the total number of mental discriminations needed to generate the program (called the program effort). Similar results were obtained by Shooman and Laemmel [5] in their study of Zipf's laws of natural languages.

2.3 Complexity as the Difficulty Experienced in Understanding a Piece of Software [6,7].

This complexity, also referred as psychological complexity, is viewed as a separate area from the quantitative computational complexity. Curtis et al. [6] describe experiments to determine which software characteristics affect understanding.

2.4 Control Flow Complexity

The geometry of the control flow graph has been studied by several workers. This graph has twin advantages over the program itself: (1) it displays the control flow graphically and thus more clearly than the program does, and (2) omits the particulars of the implementing algorithm, thus reducing the level of detail. It is believed by many (including this author) that "the secret of complexity" resides in the control flow graph, ready to be detected by a sufficiently clever detective. Up to date the researchers studied a countable property of the control graph and used the resulting number as measure of complexity.

2.4.1 The Cyclomatic Complexity Measure [2].

The cyclomatic measure gives the complexity as e-n+2p, where e is the number of edges, n the number of nodes and p the number of connected components (i.e., the number of flowcharts). For example, a program with two called subroutines (requiring one flowchart for the program and two flowcharts for the subroutines) having 13 total edges and 13 total nodes has the cyclomatic complexity

$$V = 13 - 13 + 2*3 = 6$$

For single-flowchart programs a simplification leads to $V = \pi+1$, where π is the number of decisions. Geometrically the cyclomatic complexity can be interpreted as the number of regions formed by the control flow graph.

2.4.2 Extended Cyclomatic Measures.

Myers [8] takes into account the additional complexity of compound conditions. His measure replaces the cyclomatic number by 2 numbers, with the first number being the cyclomatic number as before, and the second number being the number of conditions plus one. Then for example, the statement,

```
IF X>3 & Y<4  THEN...
              ELSE...
```

has the cyclometric complexity of 2 (because π=1 and p=1) and the extended complexity of 2:3 (because there are 2 conditions).

Basili and Reiter [9] examined four variations in determining cyclomatic complexity, involving from different weighting of CASE statements and compound predicates. The first variation is the original cyclomatic number. In the second variation each condition in a compound predicate contributes a unit to the cyclomatic number. In the third variation a compound condition still counts as a single unit, but each CASE statement of n alternatives

contributes floor[$\log_2(n)$] units[1] to the cyclomatic number (in the first 2 variations such a CASE statement contributes n units to the cyclomatic number). In the fourth variation each condition in a compound predicate is counted and the CASE statement contributes logarithmically as in the fourth variation.

2.4.3 Maximal Intersection Number (MIN) [10].

This measure is derived from the control graph by drawing a line through it and counting the maximal number of intersections (MIN). If the graph is made up of substructures in sequence, this number is obtained as the sum of MINS of all subparts - 2*number of subparts + 2.

2.4.4 The "Knots" Complexity [11].

Here we count the intersections (i.e., knots) of the control graph flowlines. Since structured programs have no intersections, they have zero knots. Hence, the knots are a measure of unstructuredness rather than complexity.

2.4.5 A Comparison of Measures of Control Flow Complexity [12]

In a recent article a comparison was made of three measures, the Halstead program effort, the cyclomatic, and the knots. The authors (A.L. Baker and S.H. Zweben) list several weaknesses of the knots. They conclude that single number characterization is inadequate, and thus more research is needed to capture the control flow complexity.

3.0 WHY IS THE PROGRAM STRUCTURE IMPORTANT?

Observe that the measures just reviewed count a property of the control flow geometry. The cyclomatic counts regions, the MIN counts maximal possible intersections, the knots counts intersections of flowlines. None of these measures take the program structure into account. For example, in the cyclometric measure $V=\pi+1$, only the number of deciders matters, and not how they are connected.

To see that connection of the deciders, that is, the structure of the program, has an important influence on complexity we have drawn in Fig. 1 two different connections of 3 deciders. Since a major part of program effort goes into testing, we argue that a program complexity should increase with test effort. To test all paths of the program segment of Fig. 1(a), four tests are needed to cover the four conditions

$$A,\ \bar{A}B,\ \bar{A}\bar{B}C,\ \bar{A}\bar{B}\bar{C}$$

The program segment of Fig. 1(b) has 8 paths. Their traversal requires the selection of eight test inputs to satisfy the eight conditons

$$ABC,\ AB\bar{C},\ A\bar{B}C,\ A\bar{B}\bar{C},\ \bar{A}BC,\ \bar{A}B\bar{C},\ \bar{A}\bar{B}C,\ \bar{A}\bar{B}\bar{C}$$

and thus more test effort than the one of Fig 1a.

---

[1]floor(x) signifies the largest integer not exceeding x

4.0 THE POLYNOMIAL MEASURE OF COMPLEXITY

The two segments of Fig. 1 lead to the conclusions:

a. each decider has 2 paths

b. the sequential connection of two deciders leads to 2^2 paths

c. more generally, the sequential connection of n deciders leads to 2^n paths

d. the parallel connection of 2 deciders results in 2+1 or 3 paths

e. the parallel connection of n deciders results in n+1 paths

We propose to denote a single decider by the variable x, as shown in Fig. 2. Then the sequential connection of two deciders is denoted by x^2 (see Fig. 3), and more generally, the sequential connection of n deciders by x^n. The power of n indicates the multiplicative nature of the program paths, created by such a connection. Thus, for example, the decider connection Fig. 3 has 2^2 program paths, while such a sequential connection of n deciders gives rise to 2^n paths.

Consider next the connection of Fig 4. This connection adds only one path to the x decider, and can be described by x+1. It is obvious now that a combination of the two basic connections of Figures 3 and 4 gives rise to polynomial expressions. As an example, Fig. 5 portrays a 2x connection, while Fig. 6 describes an x(x+1) or x^2+x situation.

These figures can be simplified by replacing each decider by a node and all sequential statements (shown by a rectangular box in the figures) by just a flowline. Fig. 7 portrays the x and x^2 situations. Similarly, the x+1 connection of Fig. 4 simplifies to the one shown in Fig. 8.

5.0 OBTAINING THE POLYNOMIAL FROM THE FLOWCHART - RULES AND EXAMPLES

Consider again the simplified flowchart of Fig. 8. If the left flowline is drawn on the right, we obtain the equivalent representations, shown in Fig. 9, all portraying the x+1 connection.

Fig. 5 can be redrawn as shown in Fig. 10, showing the addition of the two x deciders. Note that the addition requires an additional decider (that is, an additional node). More generally, if there are two flowcharts described by the polynomials $p_1(x)$ and $p_2(x)$, then the parallel connection of these two flowcharts (shown in Fig. 11) is described by $p_1(x) + p_2(x)$.

The rules for obtaining a polynomial from a flowchart can now be formulated. These are:

1. A flowchart composed from two parallel flowcharts (see Fig. 11) is described by the sum of the polynomials for each component flowchart.

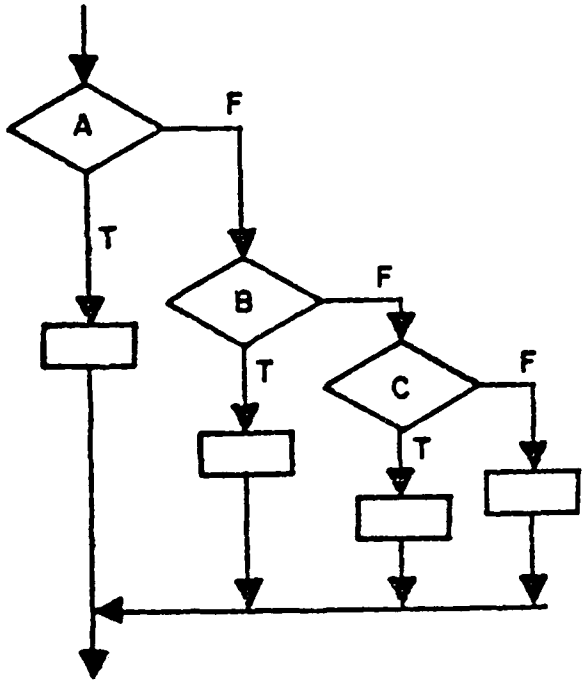


Fig. 1(a) A FOUR-TESTS CONNECTION

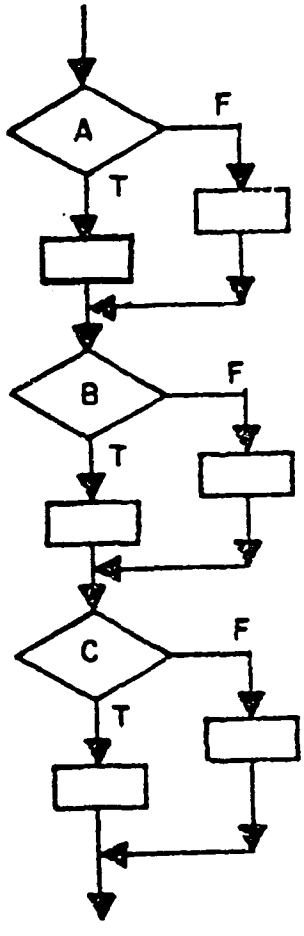


Fig. 1(b) AN EIGHT-TESTS CONNECTION

FIG. 1 TWO DIFFERENT CONNECTIONS OF THREE DECIDERS

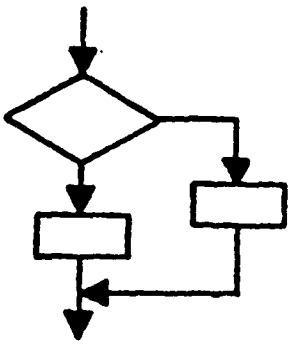


FIG. 2 A DECIDER DESCRIBED BY x

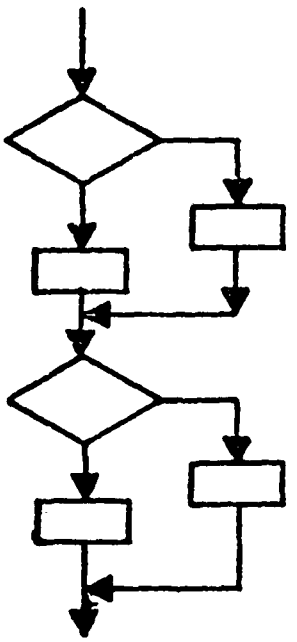


FIG. 3 AN x^2 CONNECTION

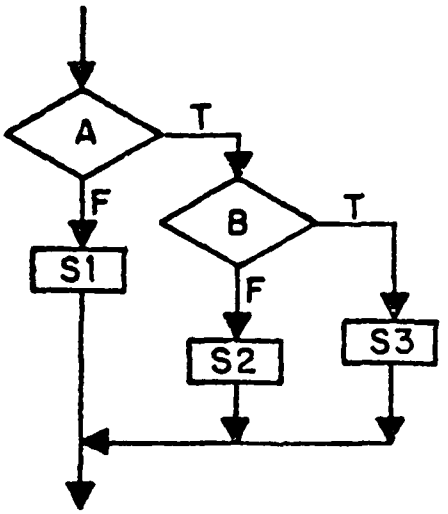


FIG. 4 AN x+1 CONNECTION

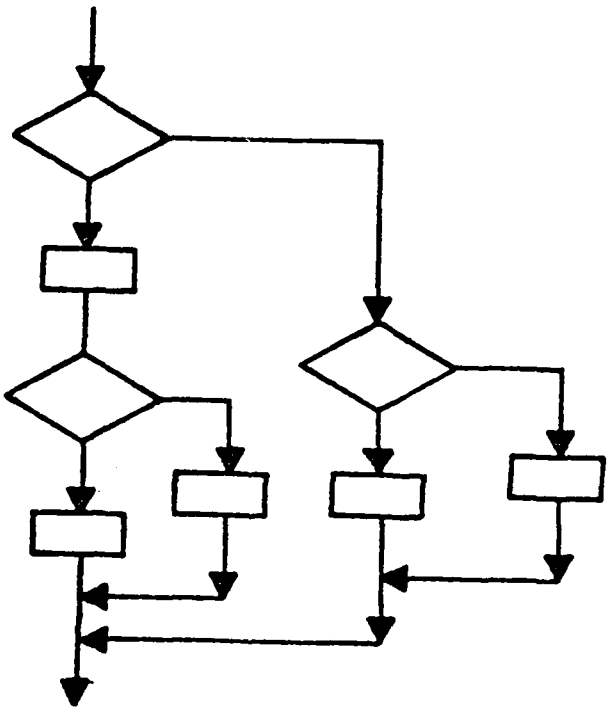


FIG. 5 A 2x CONNECTION

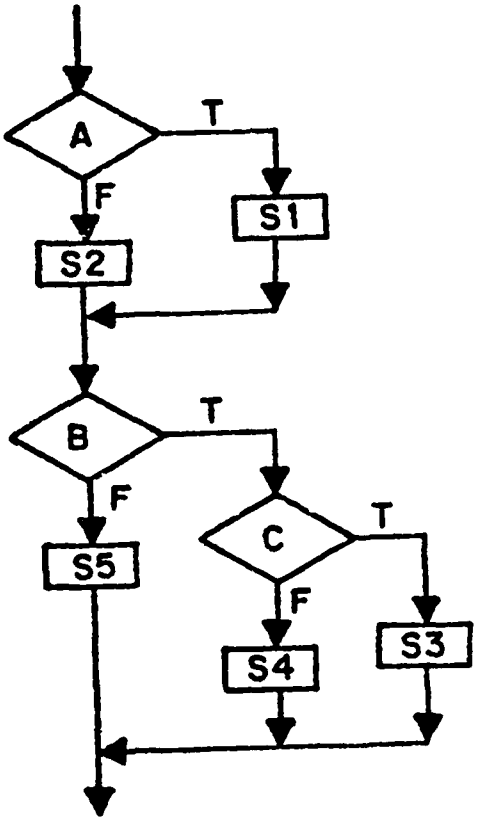


FIG. 6 AN $x^2 + x$ CONNECTION

FIG. 7(a)
SIMPLIFIED
REPRESENTATION OF
A DECIDER

FIG. 7(b)
AN x^2 CONNECTION

FIG. 7 SIMPLIFIED REPRESENTATIONS OF DECIDERS

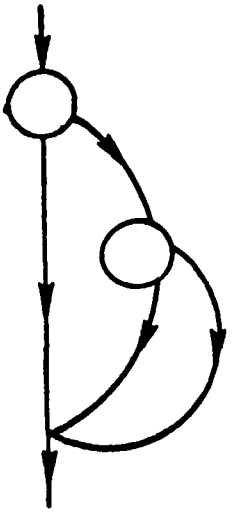


FIG. 8 SIMPLIFIED REPRESENTATION OF FIG. 4

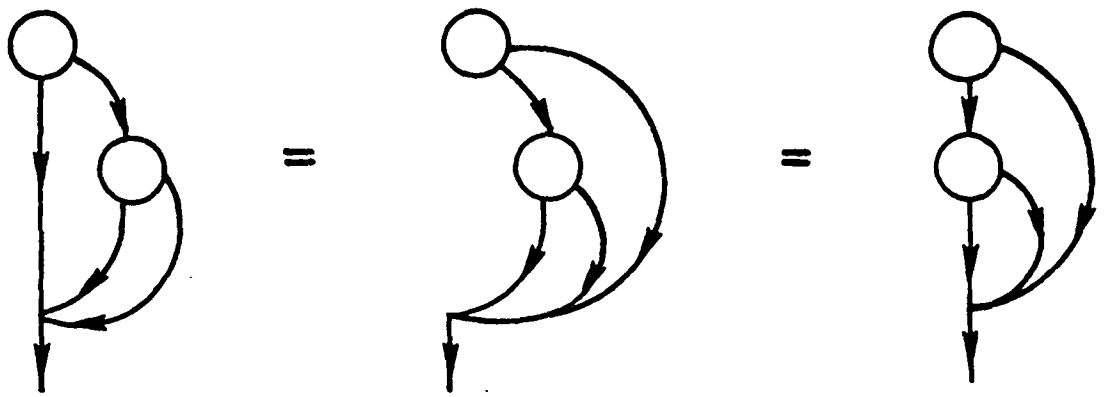


FIG. 9 MANIPULATIONS ON THE FLOWCHART OF FIG. 8

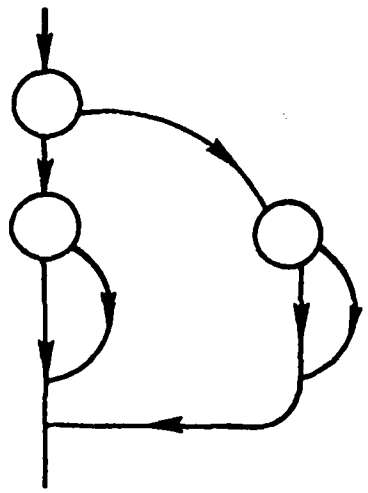


FIG. 10 THE 2x CONNECTION OF FIG. 5 REDRAWN

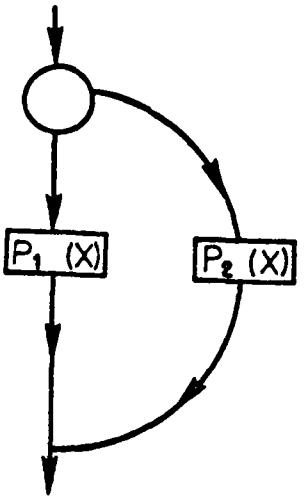


FIG. 11 FLOWCHART DESCRIBED BY $p_1(x) + p_2(x)$

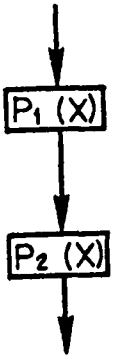


FIG. 12 FLOWCHART DESCRIBED BY $p_1(x)\ p_2(x)$

2. A flowchart composed from two sequential flowcharts is described by the product of the polynomials for each component flowchart. This situation is portrayed in Fig. 12.

3. The basic elements of a flowchart are a flowline (characterized by 1) and a decider (characterized by x), as shown in Fig. 13.

With these rules we shall obtain polynomials for several examples. For the flowchart of Fig. 14 we have the sequence of x+1 and x+2. Hence this flowchart is described by (x+1)(x+2) or x^2+3x+2. For the flowchart of Fig. 15, x is paralleled with 1 giving x+1. For the flowchart of Fig. 16, x^2 is paralleled with x giving x^2+x.

6.0 UNSTRUCTURED PROGRAMS

Up to now all the examples led to structured programs with the describing polynomial obtained from inspection of the flowchart. We will now illustrate how to obtain the polynomial for unstructured programs.

Observe that all unstructured program can be converted to a structured one by tracing the path with GO TOs, eliminating the GO TOs, and duplicating the code. As an example, consider the flowchart of Fig. 17 representing an unstructed program. Note that the conversion shown in Fig. 18 translates into the PL/I program segments:

```
IF C1 THEN DO;
                S1;
                IF C2 THEN DO;
                                S2;
                            L: S3;
                            END;
                       ELSE S5;
           END;
      ELSE DO;
                S4;
                GO TO L;
           END;
```

for the unstructured flowchart and

```
IF C1 THEN DO;
                S1;
                IF C2 THEN DO;
                                S2;
                                S3;
                            END;
                       ELSE S5;
           END;
      ELSE DO;
                S4;
                S3;
           END;
```

FIG. 13(a) THE "1" ELEMENT

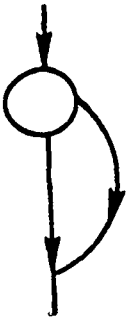


FIG. 13(b) THE "x" ELEMENT

FIG. 13 BASIC ELEMENTS OF A FLOWCHART

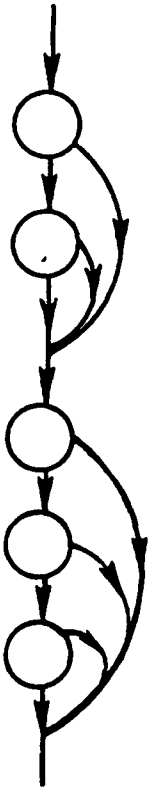


FIG. 14 AN $x^2 + 3x + 2$ CONNECTION REALIZED BY SERIES SEQUENCE OF x+1 AND x+2

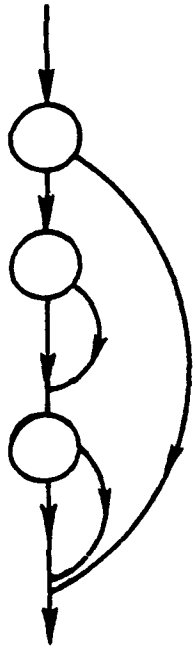


FIG. 15 AN $x^2 + 1$ CONNECTION REALIZED BY PARALLELING x^2 AND 1

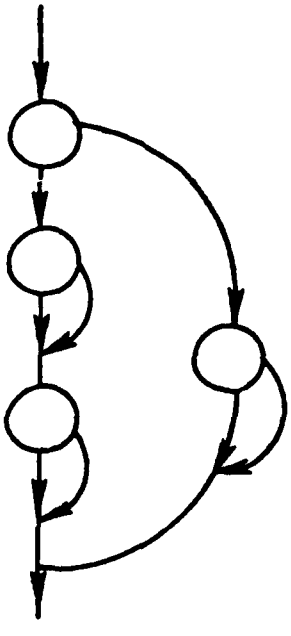


FIG. 16 AN $x^2 + x$ CONNECTION REALIZED BY PARALLELING x^2 AND x

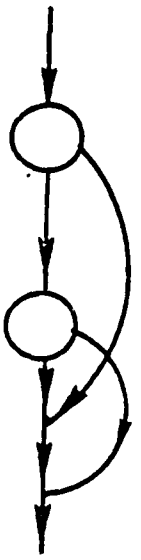


FIG. 17 FLOWCHART FOR AN UNSTRUCTURED PROGRAM

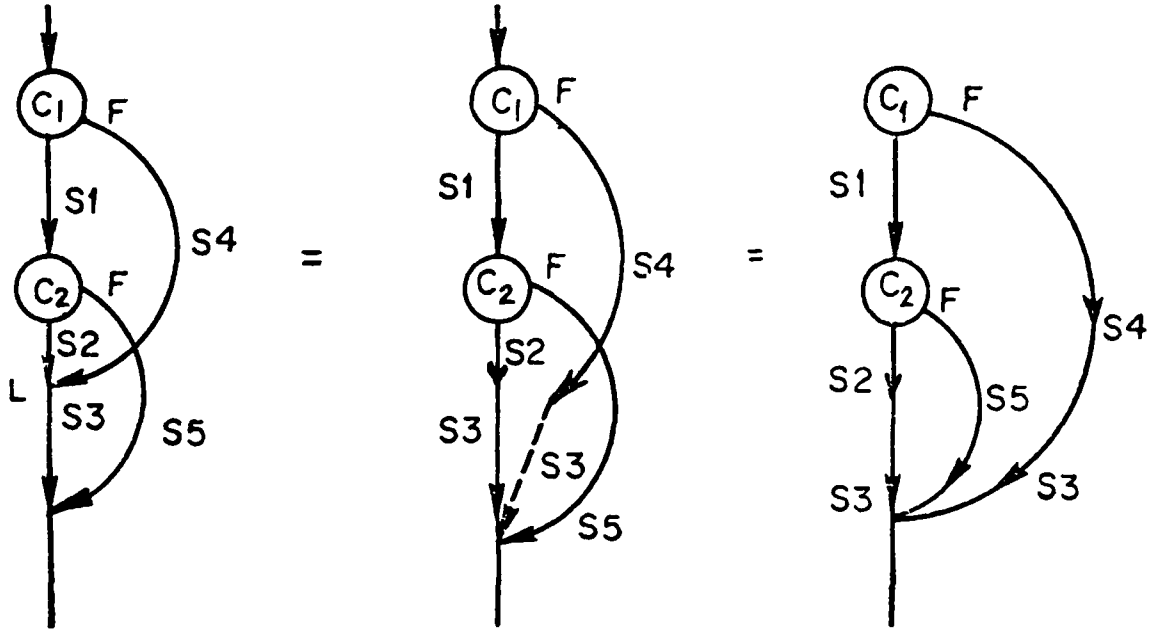


FIG. 18 CONVERSION OF AN UNSTRUCTURED PROGRAM TO A STRUCTURED ONE

for the structured flowchart. Intuitively, these two segments have the same complexity and are both described by the same polynomial. The inspection of the last flowchart of Fig. 18 reveals it to be a parallel connection of x and 1, giving x+1 for the polynomial.

As a second example consider the following PL/I program segment, illustrating how to evaluate the complexity of an unstructed code.

```
     IF A=B THEN GO TO L1;
     GO TO NO;
L1:  IF A=C THEN GO TO L2;
     GO TO NO;
L2:  PUT LIST ('SUCCESS');
     GO TO L3;
NO:  PUT LIST ('FAILURE');
L3:
```

This fragment is flowcharted in Fig. 19. Transformation to a structured program, shown in Fig. 20, shows it again to be a parallel connection of x and 1 yielding, as before, the x+1 polynomial.

As a third example, consider the flowchart of Fig. 21. As before, we want to obtain its polynomial. The simplified flowchart and its conversion to a structured form are shown in Fig. 22. The polynomial is obtained from paralleling x+1 and x resulting in 2x+1. The structured flowchart, equivalent to the one of Fig. 21, is shown in Fig. 23.

Incidentally, it may be of interest to contrast the programs for the two flowcharts of Fig. 21 and 23. For Fig. 21 the program segment (in PL/I) is:

```
IF A THEN DO;
               S1;
               GO TO L;
          END;
     ELSE IF B THEN S2;
               ELSE L:  IF C THEN S3;
                             ELSE S4;
```

while for Fig. 23 we obtain:

```
IF A THEN DO;
               S1;
               IF C THEN S3;
                    ELSE S4;
          END;
     ELSE IF B THEN S2;
                ELSE IF C THEN S3;
                          ELSE S4;
```

which are intuitively of same complexity, as asserted.

We have just illustrated how to obtain the polynomial measure of complexity of a structured or unstructured flowchart containing just sequential statements and deciders. We shall next treat flowcharts containing loops and CASE statements.

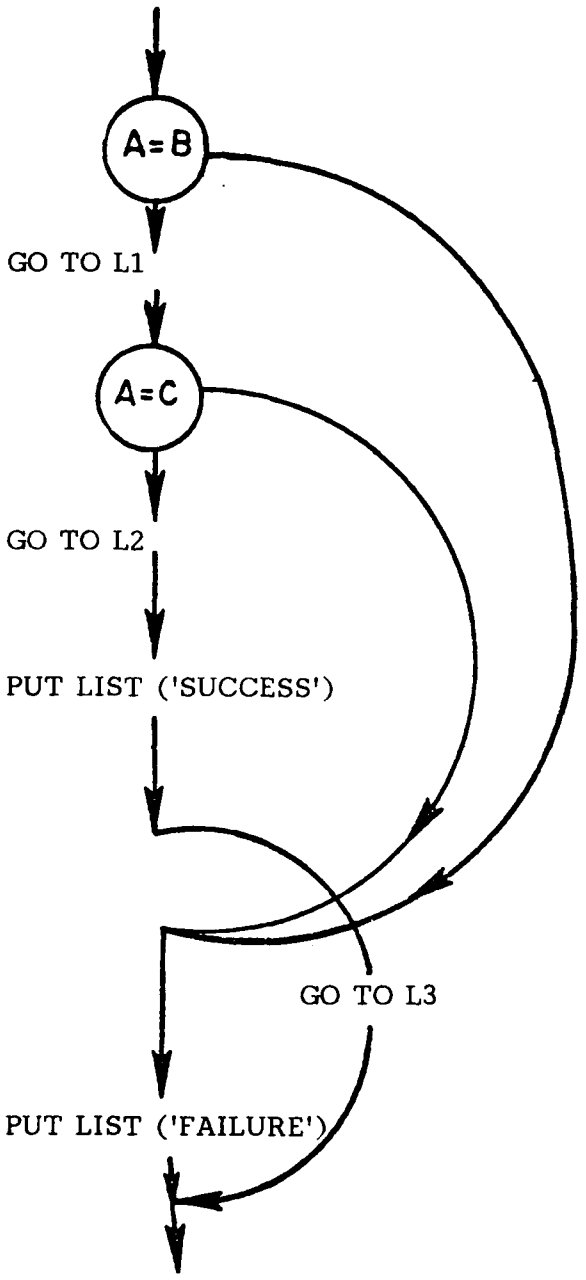


FIG. 19 FLOWCHART FOR THE PROGRAM SEGMENT OF THE SECOND EXAMPLE

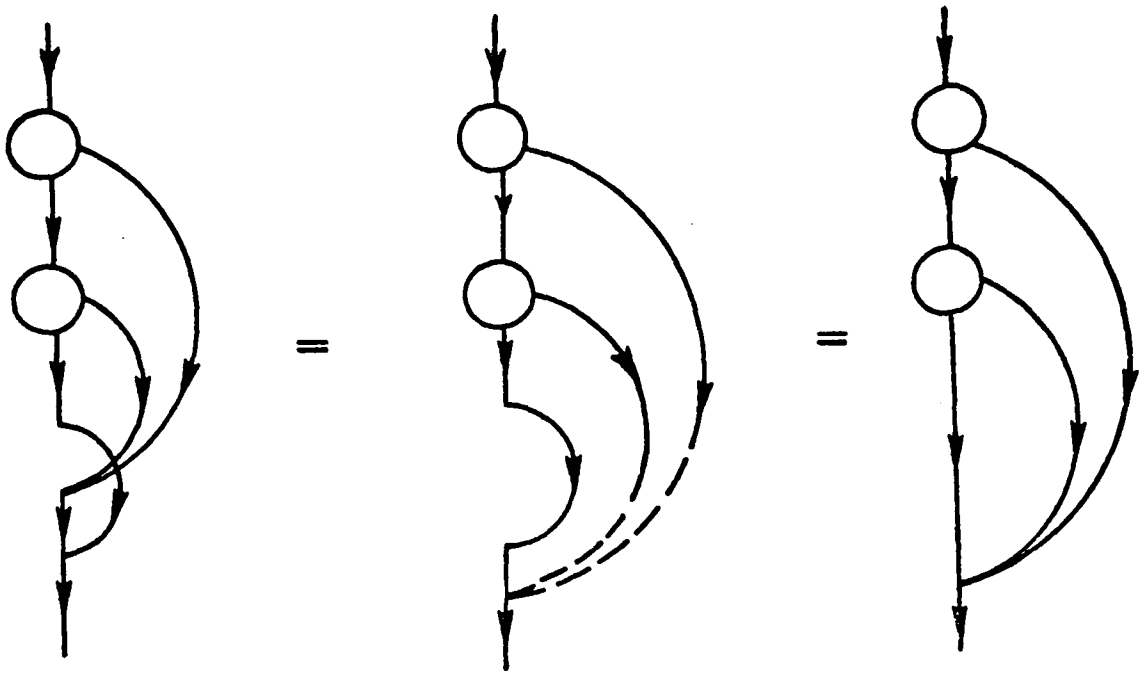


FIG. 20 CONVERSION OF THE UNSTRUCTURED FLOWCHART OF FIG. 19 TO A STRUCTURED ONE

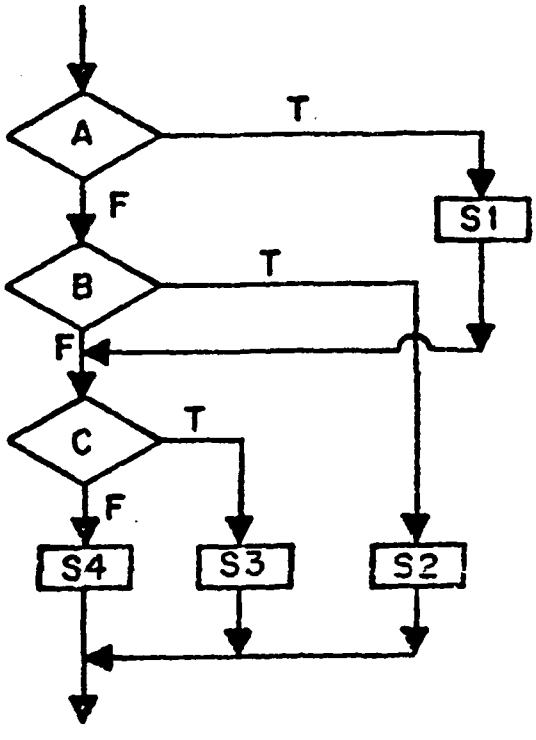


FIG. 21 FLOWCHART FOR THE THIRD EXAMPLE

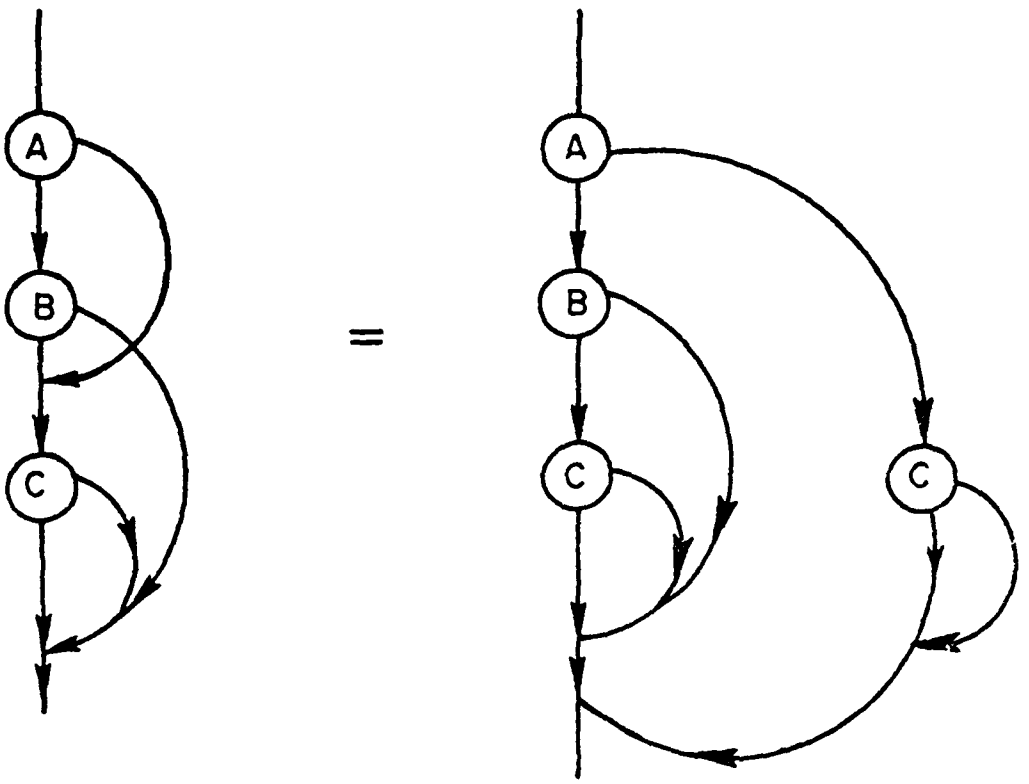


FIG. 22 SIMPLIFIED FLOWCHART AND ITS CONVERSION TO STRUCTURED FORM

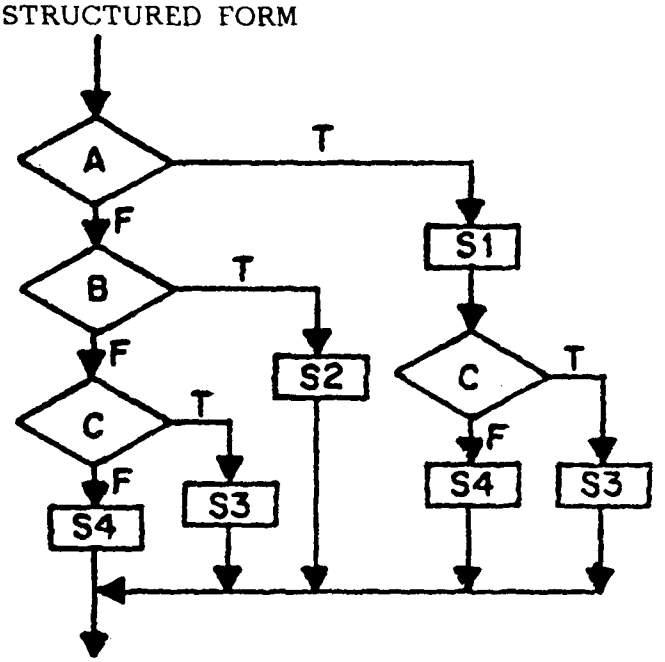


FIG. 23 A FLOWCHART EQUIVALENT TO THE ONE OF FIG. 21

7.0 REPRESENTATION OF LOOPS

Even though any program can be constructed with just sequences and deciders, such a construction is awkward without loops. Consequently, let us consider the effect of a loop on the polynomial measure p(x).

The basic representation of a loop is through the DOWHILE element shown in Fig. 24. We want to convert by geometric manipulations a flowchart with loops into a loopless one. Such a manipulation has been suggested by Shooman [13] and involves node splitting as portrayed in Fig. 25. Since the last flowchart in Fig. 25(c) is the parallel connection of x and 1, it (and thus the loop) is described by x+1.

The justification for this manipulation lies in the practice of testing a loop just twice, once for the initial value (of the control variable or control expression), and then again for the final value (i.e., the value of the control variable just before exit). Such a reduction in testing is commonplace to reduce both the volume of test data (which has to be studied) and the testing cost. For example, the loop:

```
I = 1;
DO WHILE (I<=1000);
    f;
    I = I+1;
END;
```

may in practice be tested by:

```
I = 1;
DO WHILE (I<1000);
    f;
    I = I+999;
END;
```

Another viewpoint also leads to same conclusion. A loop such as the one in Fig. 26(a) converts generally into many deciders which are in turn approximated by just two deciders, as shown in Fig. 26(c). This is justified by viewing two loops each with the same initial value and body, but with a different final value as for example,

```
DO I = 1 TO 2;
    f;
END;
```

and:

```
DO I = 1 TO 1000;
    f;
END;
```

as being of equal complexity, because they both require the same testing effort.

We have just established that a loop is described by x+1. Since loops can be either consecutive or nested, let us next obtain the polynomials for both these situations.

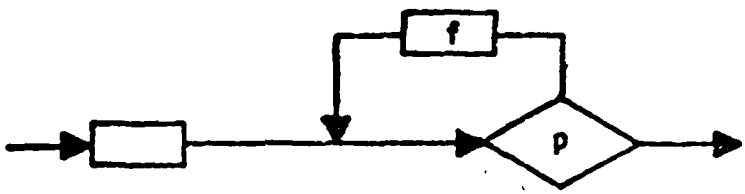
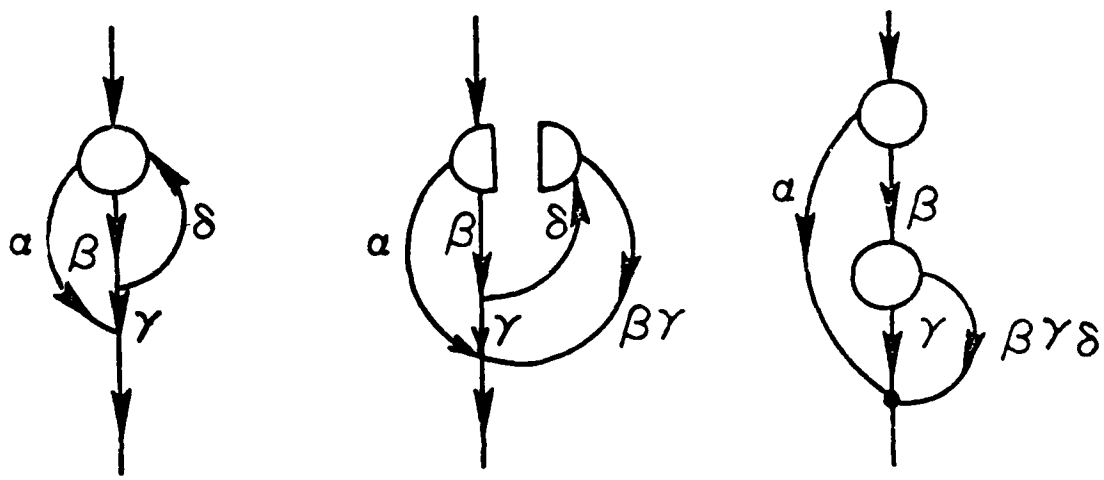


FIG. 24 THE DOWHILE ELEMENT

(This element portrays the program segment
Statement;
DO WHILE (p);
f;
END;)

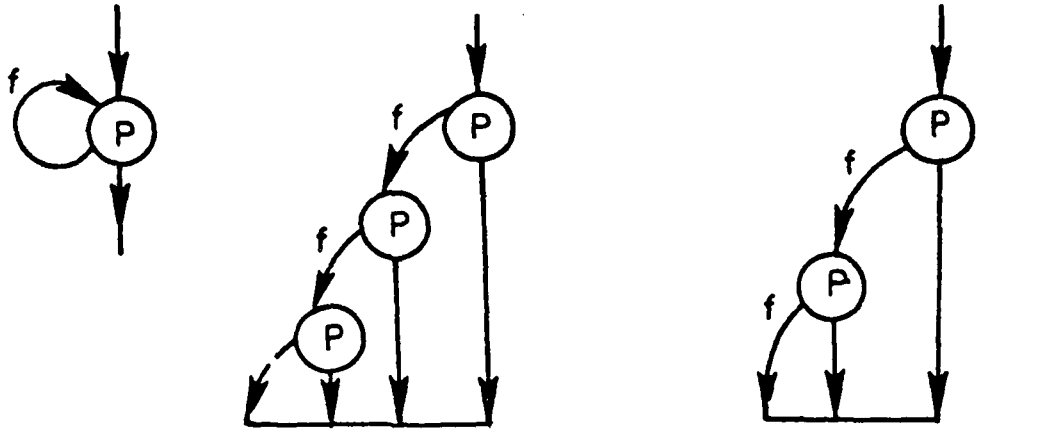


(a) ORIGINAL LOOP

(b) SPLITTING THE NODE

(c) EQUIVALENT LOOPLESS FLOWCHART

FIG. 25 THE CONVERSION OF LOOP TO AN EQUIVALENT LOOPLESS FLOWCHART



(a) THE ORIGINAL LOOP

(b) COMPLETE CONVERSION

(c) REDUCED CONVERSION (described by x+1)

FIG 26 REDUCED CONVERSION OF A LOOP

(a) TWO CONSECUTIVE LOOPS

(b) EQUIVALENT FLOWCHART

FIG. 27 TWO CONSECUTIVE LOOPS AND THEIR EQUIVALENT FLOWCHART

Two consecutive loops are shown in Fig. 27(a). Since each loop is described by x+1 and this is a sequential connection of two such x+1 polynomials, the polynomial for both these loops is $(x+1)^2$ (see Fig. 12). More generally, n consecutive loops results in the $(x+1)^n$ polynomial.

When a loop is nested within another, we have the situation portrayed in Fig. 28(a). As before, the nested inner loop may be replaced by x+1, (that is, its polynomial), resulting in the flowchart of Fig. 28(b). But this flowchart is the same as the one of Fig. 26(a), with f being replaced by x+1, as shown in Fig. 28(c). This leads to the polynomial $p_2(x)$ for the nesting of 2 loops as

$$p_2(x) = \underbrace{[(x+1) + 1]}_{\substack{\text{polynomial} \\ \text{for the first} \\ \text{decider}}} (x+1) + 1 = (x+2)(x+1) + 1$$

Again we can generalize to a nesting of n loops and obtain $p_n(x) = [p_{n-1}(x) + 1]\, p_{n-1}(x) + 1$. Specific examples for n=2,3, and 4 are

$$p_2(x) = x^2 + 3x + 3$$

$$p_3(x) = (x^2+3x+4)(x^2+3x+3) + 1 = x^4+6x^3+16x^2+21x+13$$

$$p_4(x) = (x^4+6x^3+16x^2+21x+14)(x^4+6x^3+16x^2+21x+13) + 1$$

Observe how quickly the polynomial grows with levels of nesting indicating the rapid increase in complexity.

We have just described how to treat loops. We shall next consider the multiple selectors (i.e., the CASE statement).

8.0 THE CASE STATEMENT

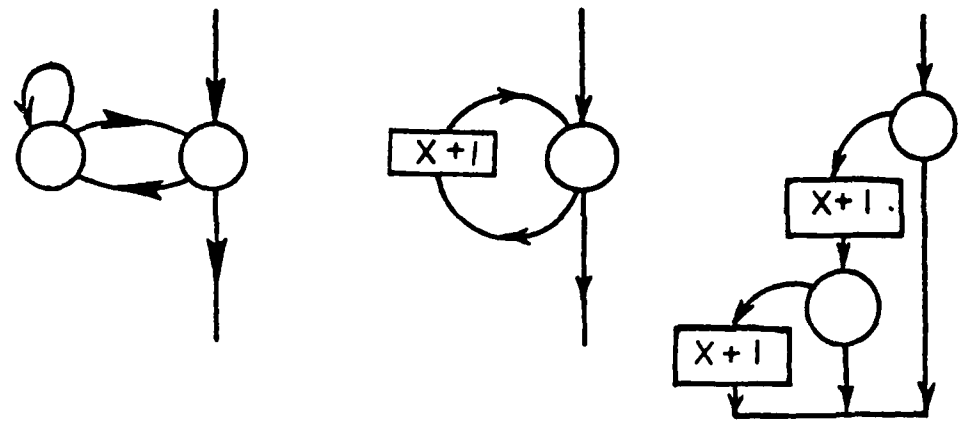
Figure 29(a) portrays a multiple selector implementable by the CASE statement with 3 cases. Using the node splitting technique as in Fig. 25, we obtain the equivalent flowchart of Fig. 29(b). This last flowchart is a parallel connection of the x and 1 elements, yielding x+1. More generally a CASE statement with n+2 cases is described by x+n.

We have just shown how to characterize a flowchart by a polynomial. We shall next consider the topic of equivalent flowcharts.

9.0 EQUIVALENT FLOWCHARTS

9.1 Definition and Examples

We will consider two flowcharts to be equivalent if they are characterized by the same polynomial. Because a polynomial can generally be partitioned into simpler terms in several ways, the different realizations lead to equivalent flowcharts.

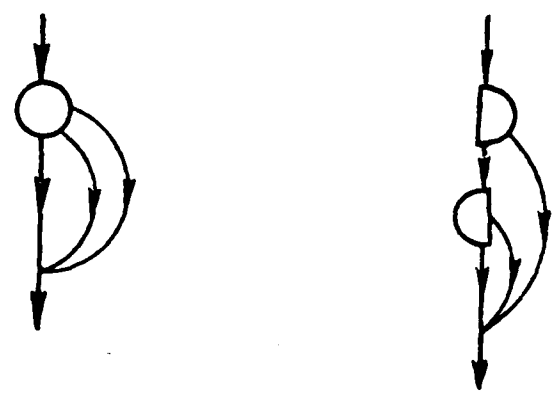


(a) A NESTED LOOP

(b) REPLACEMENT OF THE INNER LOOP BY x+1

(c) EQUIVALENT FLOWCHART

FIG. 28 A NESTED LOOP AND ITS EQUIVALENT FLOWCHART



(a) A MULTIPLE SELECTOR

(b) EQUIVALENT FLOWCHART

FIG. 29 A MULTIPLE SELECTOR AND ITS EQUIVALENT FLOWCHART

To illustrate different realizations consider Fig. 6, portraying the x^2+x connection. In Fig. 6 the bottom part represents x+1 and the top decider an x multiplier, resulting in $x(x+1)$ or x^2+x. It is also possible to realize x^2+x by paralleling x^2 and x, as shown in Fig. 30. Note that the first (i.e., top) decider separates x^2 form x.

Observe that the flowchart of Fig. 30 has one more decider than the flowchart of Fig. 6. Comparing the program segment for Fig. 30, that is,

```
IF A THEN IF B THEN S1;
                ELSE S2;
     ELSE DO;
             IF  C THEN S3;
                    ELSE S4;
             IF  D THEN S5;
                    ELSE S6;
          END;
```

with the program segment for Fig. 6, i.e.,

```
IF A THEN S1;
     ELSE S2;
IF B THEN IF C THEN S3;
                ELSE S4;
     ELSE S5;
```

We ask whether the definition agrees with our intuitive notion of complexity. If analyzing the first segment (for Fig. 30), we deduce that:

S1 is executed for AB

S2 is executed for $A\bar{B}$

S3 and S5 are executed for $\bar{A}CD$

S3 and S6 are executed for $\bar{A}C\bar{D}$

S4 and S5 are executed for $\bar{A}\bar{C}D$

S4 and S6 are executed for $\bar{A}\bar{C}\bar{D}$

The analysis of the second segment (for Fig. 6) tells that:

S1 and S3 are executed for ABC

S1 and S4 are executed for $AB\bar{C}$

S1 and S5 are executed for $A\bar{B}$

S2 and S3 are executed for $\bar{A}BC$

S2 and S4 are executed for $\bar{A}B\bar{C}$

S2 and S5 are executed for $\bar{A}\bar{B}$

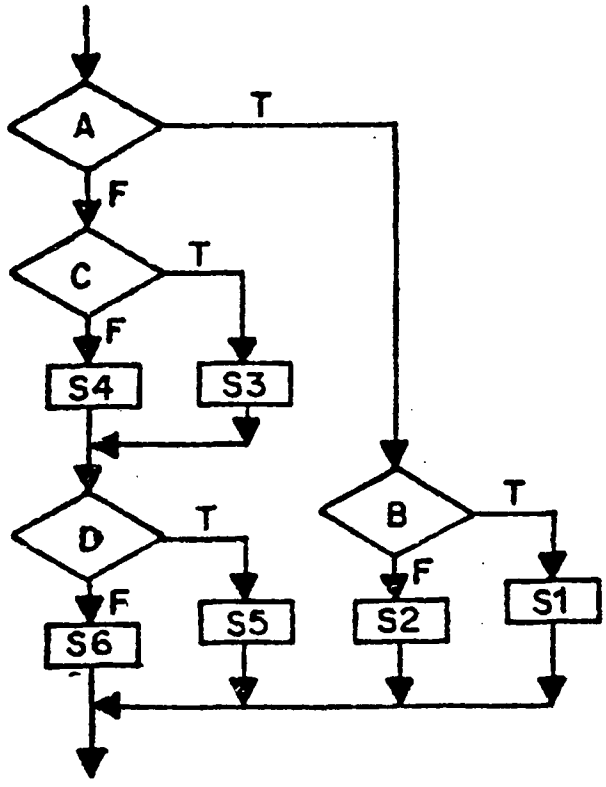


FIG. 30 AN x^2+x CONNECTION REALIZED BY PARALLELING x^2 AND x

which indicates the same testing is needed to verify it as the first segment, and thus intuitively of the same complexity as the first segments.

An additional example of equivalent flowcharts is shown in Figs. 31 and 32. In Fig. 31 x^2+2x+3 is realized as an additive connection of x^2, x, and x+3. We shall call such a partitioning of a polynomial as "additive" partitioning. In Fig. 32 x multiplies x+2 and then 3 is added. We shall call this second type of partitioning "multiplicative" partitioning. The analysis of program segments arising from both constructions leads to the conclusion that both will require the same testing effort and thus have the same complexity (see Appendix).

9.2 Realization of the Polynomial with Fewest Deciders

It is evident from the last 2 examples that additive partitioning leads to a realization with more deciders than does multiplicative partitioning. As an example consider the different realizations of x^3+3x^2+x+2.

1. Additive partitioning:

   x^3 requires 3 deciders

   $3x^2$ requires 3*2+2=8 deciders (2 extra deciders needed to add $x^2+x^2+x^2$)

   x requires 1 decider

   $\therefore$ x^3+3x^2+x requires 16 deciders (2 additional deciders for the 2 additions)

and finally

   x^3+3x^2+x+2 requires 16 deciders (2 deciders for +2)

2. $x^2(x+3)$ + (x+2) partitioning

   x^2 requires 2 deciders

   x+3 requires 4 deciders

   $x^2(x+3)$ requires 6 deciders

   x+2 requires 3 deciders

   $\therefore$ $x^2(x+3)$ + (x+2) requires 10 deciders (one additional decider for the addition).

3. x(x(x+3)+1) + 2 partitioning

   x+3 requires 4 deciders

   x(x+3) requires 5 deciders

   x(x+3)+1 requires 6 deciders

   x(x(x+3)+1) requires 7 deciders

   x(x(x+3)+1)+2 requires 9 deciders

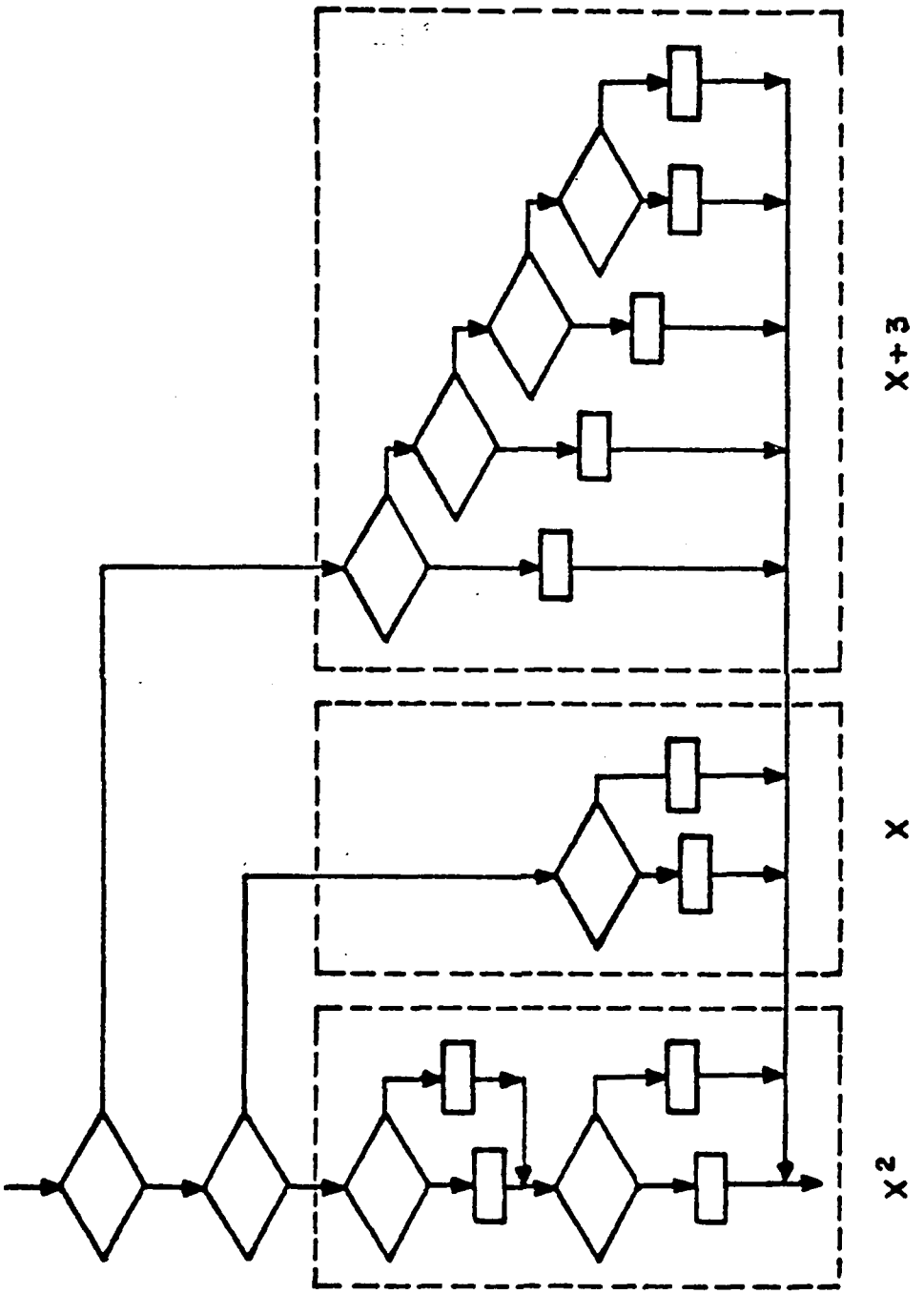


FIG. 31 x^2+2x+3 REALIZED AS A SUM OF $(x^2)+(x)+(x+3)$

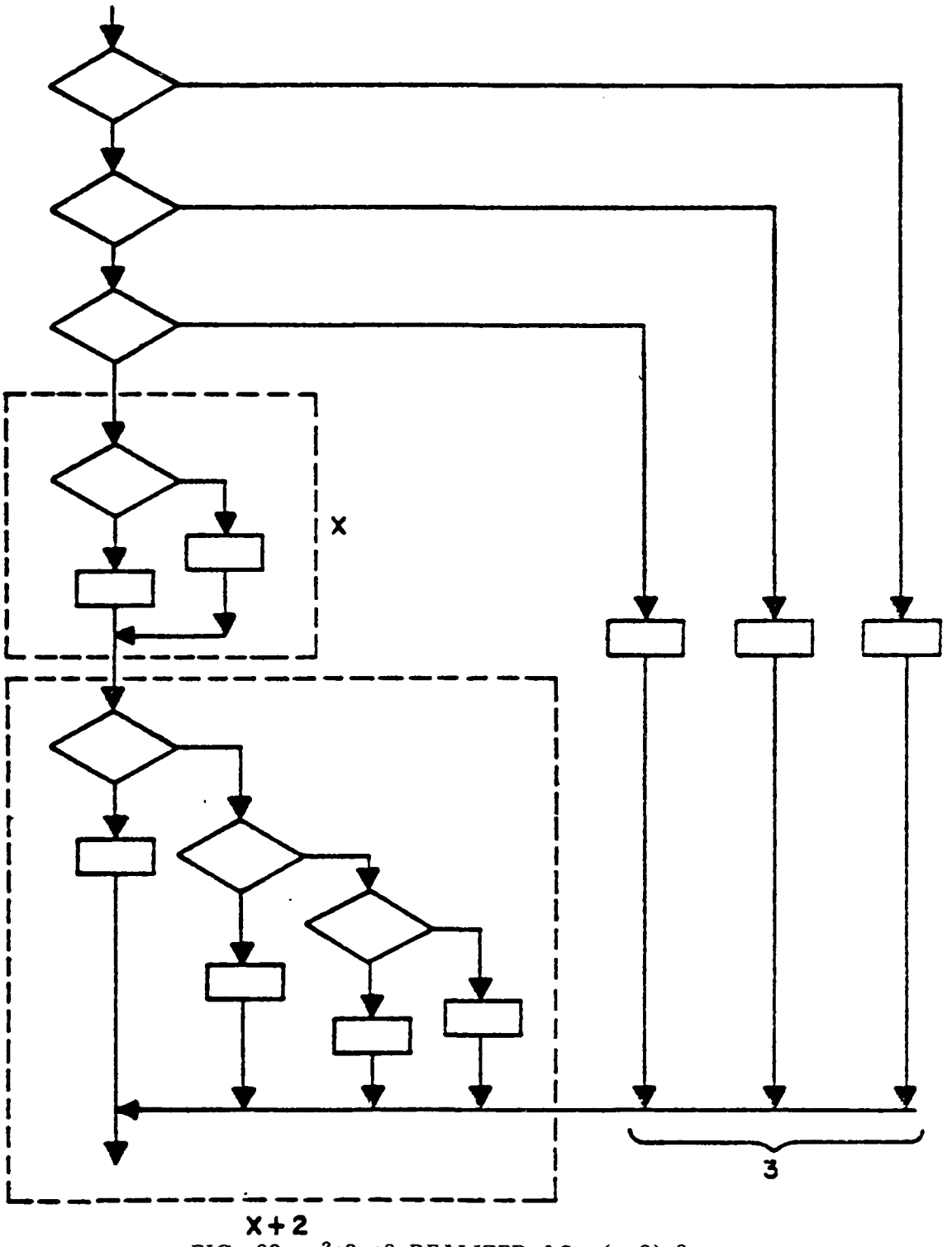


FIG. 32 x^2+2x+3 REALIZED AS $x(x+2)+3$

From this example it can be deduced that a minimal decider realization of a polynomial will be obtained if the polynomial is expressed in the "nested from" as in the last partitioning. This is so because no extra deciders are needed for addition. Then for the general cubic polynomial

$$a_3x^3 + a_2x^2 + a_1x + a_0 = a_0+x(a_1+x(a_2+a_3x))$$

(where obviously all a_i are non-negative integers) we need

$2a_3$-1 deciders for a_3x (one decider for each of the a_3 x and a_3-1 deciders for their additions)

a_2+2a_3-1 deciders for a_2+a_3x

a_2+2a_3 deciders for $x(a_2+a_3x)$

$a_1+a_2+2a_3$ deciders for $a_1+x(a_2+a_3x)$

$a_1+a_2+2a_3+1$ deciders for $x(a_1+x(a_2+a_3x))$

$a_0+a_1+a_2+2a_3+1$ deciders for the polynomial

This result can be generalized with the theorem

Theorem

The minimal decider realization of the polynomial

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

requires

$$n-1 + 2a_n - 1 + \sum_{k=0}^{n-1} a_k = n-2 + 2a_n + \sum_{k=0}^{n-1} a_k \qquad (9.1)$$

deciders. This realization is achieved with the polynomial expressed in the so called "nested form."

9.3 The Deciders Needed for Additive Partitioning

We have just determined the lower bound on deciders realizing a polynomial. Let us now determine the number of deciders needed for additive partitioning. This will give us an upper bound on the deciders. Consider again the polynomial $a_3x + a_2x^2 + ax + a_0$.

a_3x^3 requires $4a^3$ - 1 deciders (a_3 - 1 deciders needed for the additions

a_2x^2 requires $3a_2$-1 deciders

a_1x requires $2a_1$ - 1 deciders

$a_3x^3+a_2x^2+a_1x$ requires $4a_3+3a_2+2a_1x-1$ deciders (2 additional deciders needed for the 2 additions)

$a_3x^3+a_2x^2+a_1+a_0$ requires $4a_3+3a_2+2a_1+a_0-1$ deciders.

Thus in general, the additive realization of the polynomial

$$p(x) = \sum_{k=0}^{n} a_k x^k$$

requires:

$$\sum_{k=0}^{n} (k+1)a_k - 1 \qquad (9.2)$$

deciders. This is the upper bound on the deciders.

From the results of (9.1) and (9.2) it follows that a polynomial p(x) may be realized with π deciders where

$$n-2 + 2a_n + \sum_{k=0}^{n-1} a_k \leq \pi \leq \sum_{k=0}^{n} (k+1)\, a_k - 1$$

10.0 APPLICATIONS OF THE MEASURE

10.1 Cyclomatic Complexity

The cyclomatic complexity [2] of a single flowchart (i.e., p=1, meaning that the program has no procedures) was given as π+1, where π is the number of deciders. From the preceeding section we have a bound on π. Consequently, a flowchart characterized by the polynomial p(x) has the cyclomatic complexity V bounded by

$$n-1 + 2a_n + \sum_{k=0}^{n-1} a_k \leq V \leq \sum_{k=0}^{n} (k+1)a_k$$

As an example, the flowchart described by

$$p(x) = 3x^2 + 2x + 1$$

has the cyclomatic complexity V bounded by

$$2\text{-}1 + 2\cdot3 + 2+1 \leq V \leq 3\cdot3 + 2\cdot2 + 1$$

or

$$10 \leq V \leq 14$$

10.2 Number of Program Paths

One of the usual tests of a program is the type 2 test [14]. In such test each program path is executed at least once. A measure of extent of such a test (and thus of test effort) is the number of program paths, which is given by

Number of program paths = p(2)

To see that this is so, consider first the two basic connections of two flowcharts, parallel and sequential. For the parallel connection (shown in Fig. 11) the paths just add, hence

> Total number of paths = paths in the flowchart described by $p_1(x)$ + paths in the flowchart described by $p_2(x)$.

For the sequential connection (shown in Fig. 12), the paths multiply. This can be noted from Fig. 33, hence,

> Total number of paths = (paths in the flowchart described by $p_1(x)$) (paths in the flowchart described by $p_2(x)$).

Since all flowcharts are comprised by either parallel or sequential connection of the two basic elements, the "1", (which has 1 path) and "x" (which has 2 paths) shown in Fig. 13, the total number of program paths of a flowchart described by p(x) is indeed p(2).

11.0 MODULAR PROGRAMS

The principal tool in the design of large software system is to divide the system into smaller units. Such a division intuitively reduces the overall complexity by allowing us to deal with smaller, less complex "modules." Two questions may be posed:

1. What effect has the modular division on the polynomial complexity?

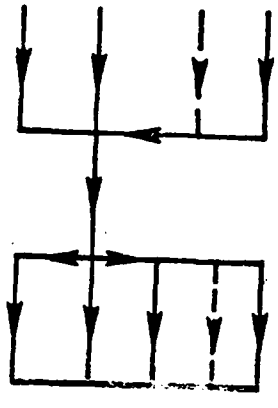
and

2. What is the optimum division, that, how should the program be divided?

Let us consider the first question. Evidently, a program with a single procedure is represented by a single flowchart, leading to a single polynomial. Similarly, a modular program consists of several procedures (one for each module), with each procedure represented by a separate flowchart. Each flowchart is characterized by its own polynomial. The entire modular program is the sum of the individual polynomials. Thus, for a modular program with n modules,

$$p(x) = \sum_{i=1}^{n} p_i(x)$$

where $p_i(x)$ is the polynomial characterizing the flowchart for the ith module.



M PATHS
[FLOWCHART DESCRIBED BY $p_1(x)$]

N PATHS
[FLOWCHART DESCRIBED BY $p_2(x)$]

FIG. 33 SEQUENTIAL CONNECTION OF TWO FLOWCHARTS, SHOWING M·N TOTAL PATHS

As an example, consider the flowchart of Fig. 32, with the decider labeled x removed to a separate flowchart. This situation is shown in Fig. 34. Fig. 34(a) is characterized by (x+2)+3 or x+5. Fig. 15(b) is characterized by x. Hence, the entire modular program is characterized by x+5+x or 2x+5, which results in a polynomial of lower complexity than the one characterizing the single flowchart of Fig. 32.

Consider next the flowchart of Fig. 31. We will now replace the two deciders labeled x^2 by a single module. This results in the two flowcharts portrayed in Fig. 35. The polynomial describing the flowchart of Fig. 35(a) is (x+1)+(x+3) or 2x+4. The module is represented by x^2. Hence, the entire modular program is characterized by x^2+2x+4 which is certainly not simpler than the original polynomial x^2+2x+3.

The observation of these two examples leads to the following conclusions:

1. In the first example a multiplicative part was replaced by a module. This had the effect of replacing a multiplicative part by an additive part. In general, if the original polynomial $p_0(x)$ can be decomposed into

   $$p_0(x) = p_1(x)\ p_2(x) + p_3(x)$$

   and $p_2(x)$ is replaced by a module, the new characterization $p_n(x)$ becomes

   $$p_n(x) = p_1(x) + p_3(x) + p_2(x)$$

   which leads to a polynomial of a lower degree and thus lower complexity.

2. In the second example an additive component was replaced by a module. If the original polynomial $p_0(x) = p_1(x) + p_2(x)$, we will let $p_2(x)$ be replaced by a module. This results in two flowcharts, one characterized by $p_1(x)+1$ and the other by $p_2(x)$, resulting in the new overall characterization of

   $$p_n(x) = p_1(x) + p_2(x) + 1$$

   and thus of no lower complexity than the original polynomial $p_0(x)$.

The above answers the first question posed in the beginning of this section, by showing the two possible effects of modular division on polynomial complexity. It also allows to answer the second question, asking how modules are to be chosen. The answer is that multiplicative terms should be replaced by modules, because such modularization lowers the degree of the polynomial p(x). The replacement of additive terms by modules is not recommended because such modularization does not lead to lower complexity.

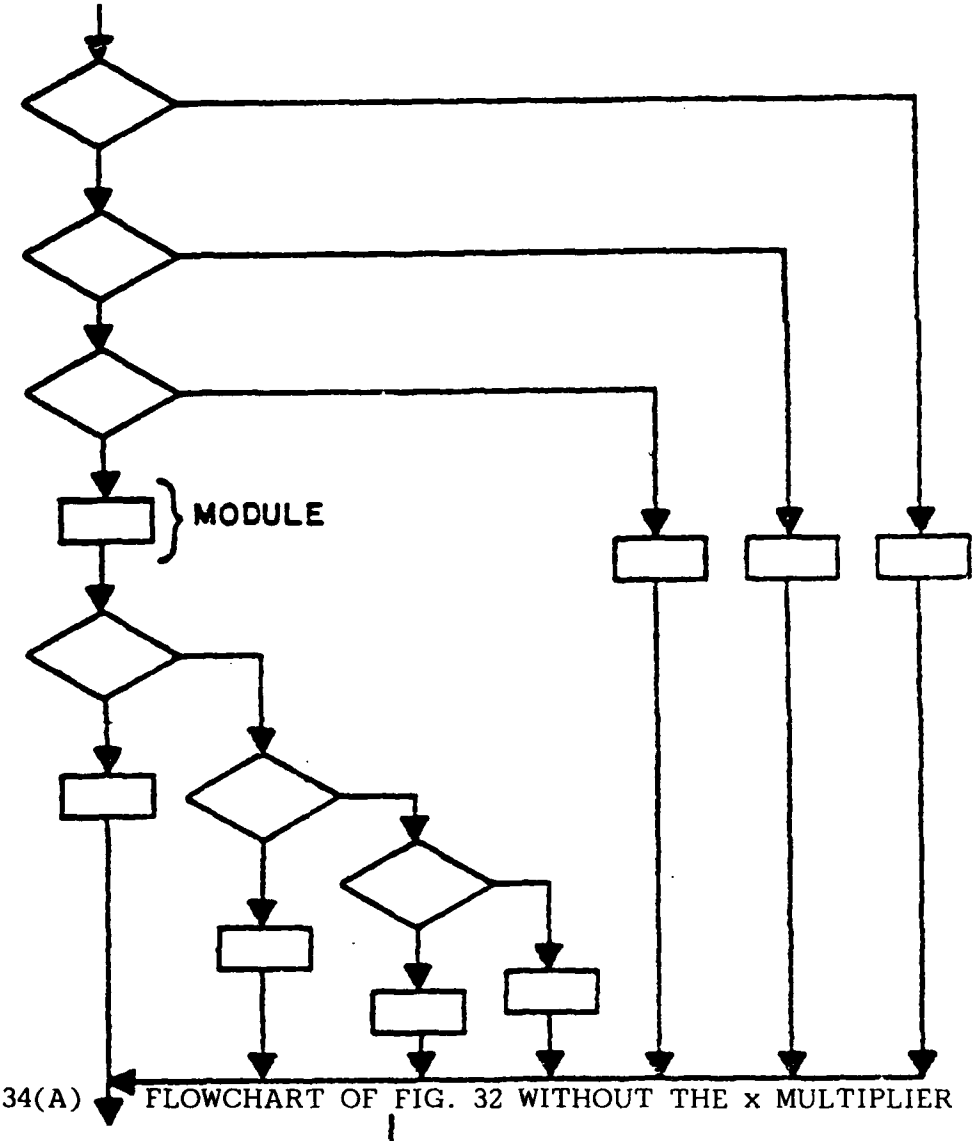


FIG. 34(A) FLOWCHART OF FIG. 32 WITHOUT THE x MULTIPLIER

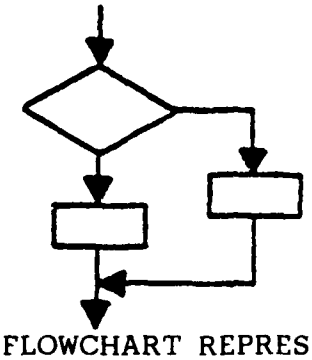


FIG. 34(B) FLOWCHART REPRESENTING x

FIG. 34 FLOWCHART OF FIG. 32 WITH THE MODULE x SHOWN SEPARATELY

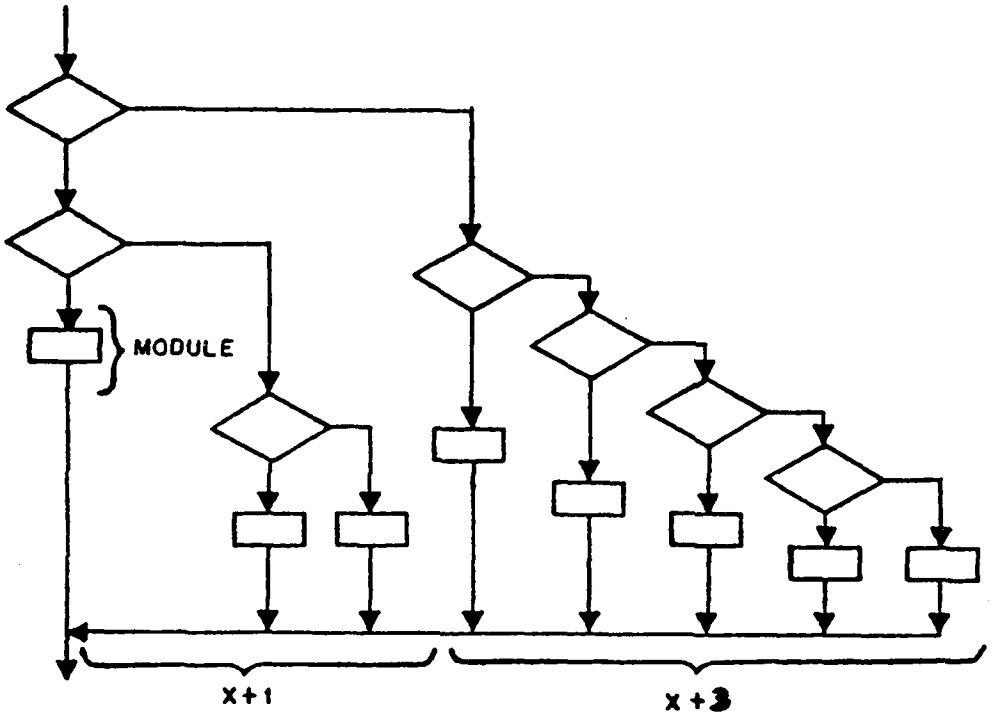


FIG. 35(A) FLOWCHART OF FIG. 31 WITHOUT THE x^2 PART

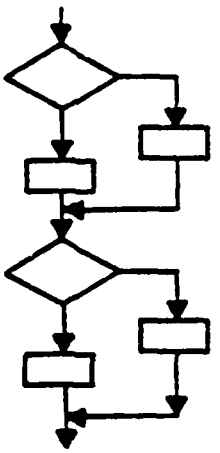


FIG. 35 (B) FLOWCHART REPRESENTING x^2

FIG. 35 FLOWCHART OF FIG. 31, *WITH THE MODULE* x^2 *SHOWN* SEPARATELY

In Section 9.0 we described the construction of a flowchart from the polynomial p(x). The construction methods given there resulted in a single flowchart. Observe that modular realizations give rise to additional construction techniques. The development of such techniques is left to the future.

12.0 ASSESSMENT OF THE POLYNOMIAL COMPLEXITY MEASURE

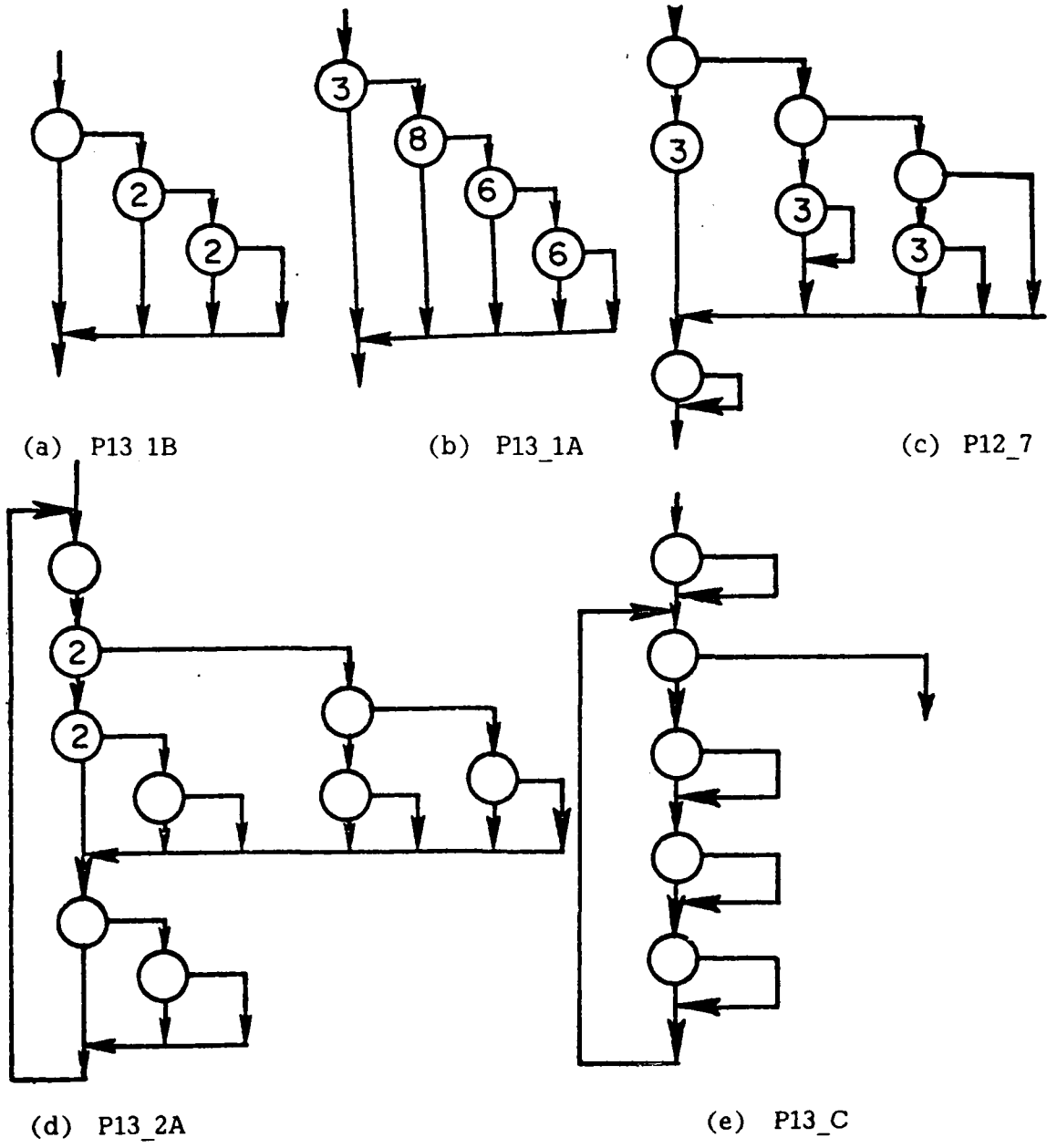
A question of interest is: "How does the polynomial measure compare with the other measures in predicting program complexity?" To answer this question, seven exercises were selected from a textbook on programming [15]. For each of these, a program was constructed, debugged, and tested, and the completion time and number of statements were recorded. The simplified flowcharts were then drawn for the programs, and are shown in Fig. 36. A number n inside a decider indicates a decider with n conditions (and no number indicates a single condition). The names P13_1B, P13_1A, and so on, are the names of the programs (and the exercises). The results are tabulated in Table 1 and include the average number of minutes expended on each statement and the four complexity measures: the cyclomatic, extended (Myers, [8]), MIN, and polynomial. The MIN measure is only defined for loopless flowcharts, thus, only the first 3 programs show this measure. The other control flow complexity measures are not shown because the programs written for the seven exercises were in PL/I, and hence did not contain the CASE statement. Thus the third and fourth variations of Basili and Reiter [9] were of no interest. (The first and second variations are the cyclomatic and the second number of the extended, respectively.) The knots are not shown because all programs are structured and hence have no knots.

The table demonstrates that polynomial complexity correlates with the effort expended on each statement, and does so better than the other measure (or the number of paths). It is interesting to note that the fifth degree polynomial with two terms in P13_1C leads to a lower "complexity" than the fourth degree polynomial with all five terms in P9_5.

13.0 SUMMARY

A new measure of complexity was introduced, one which considers deciders and the way they are connected. The measure is unique and interprets a program flowchart through a polynomial. It was shown that the measure yields a bound on cyclomatic complexity, and the expected number of path tests. The measure allows the comparison of alternate designs and the choice of the design of minimal complexity. A further advantage of the measure is its application to modular designs; the measure tells which modularization contributes to a reduction in over complexity.

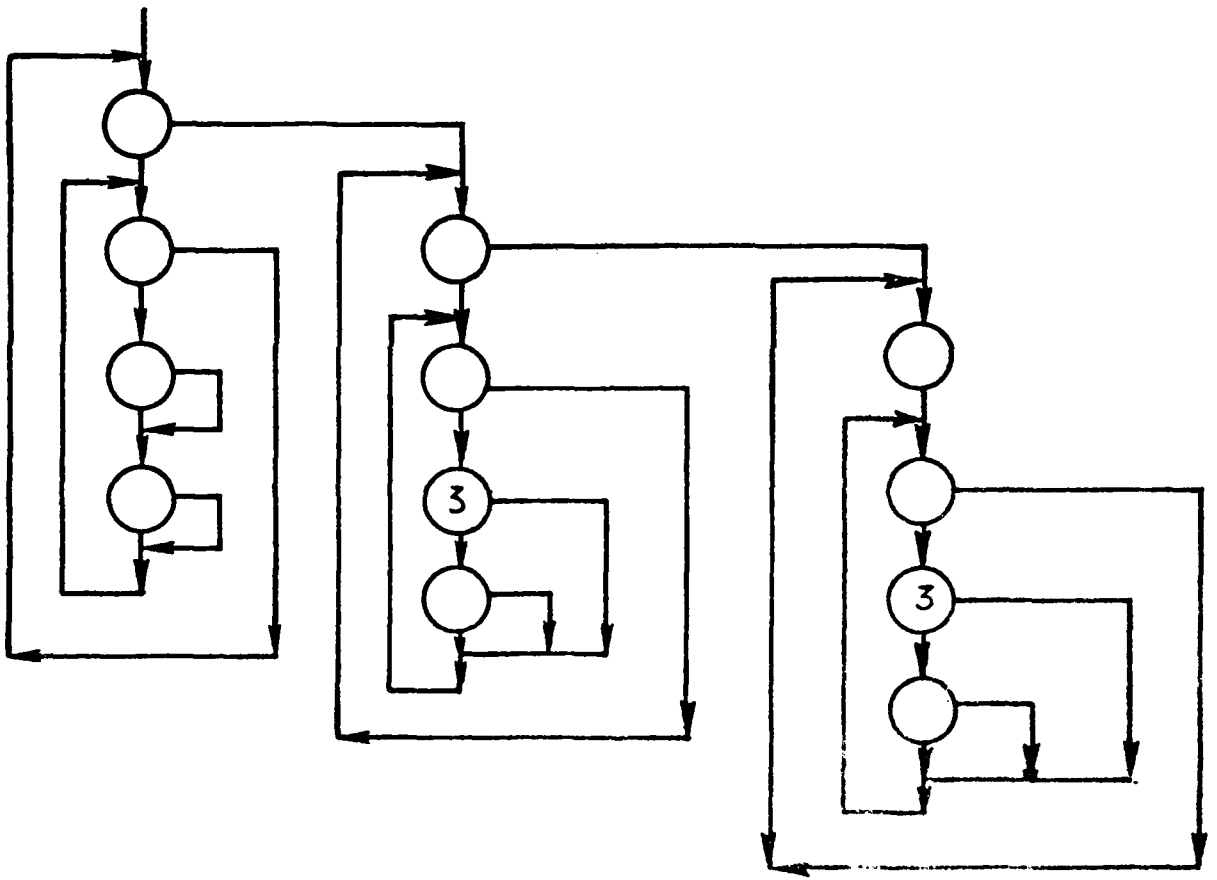
The report concludes with a comparison between the polynomial measure and several other popular measures.



(a) P13 1B (b) P13_1A (c) P12_7

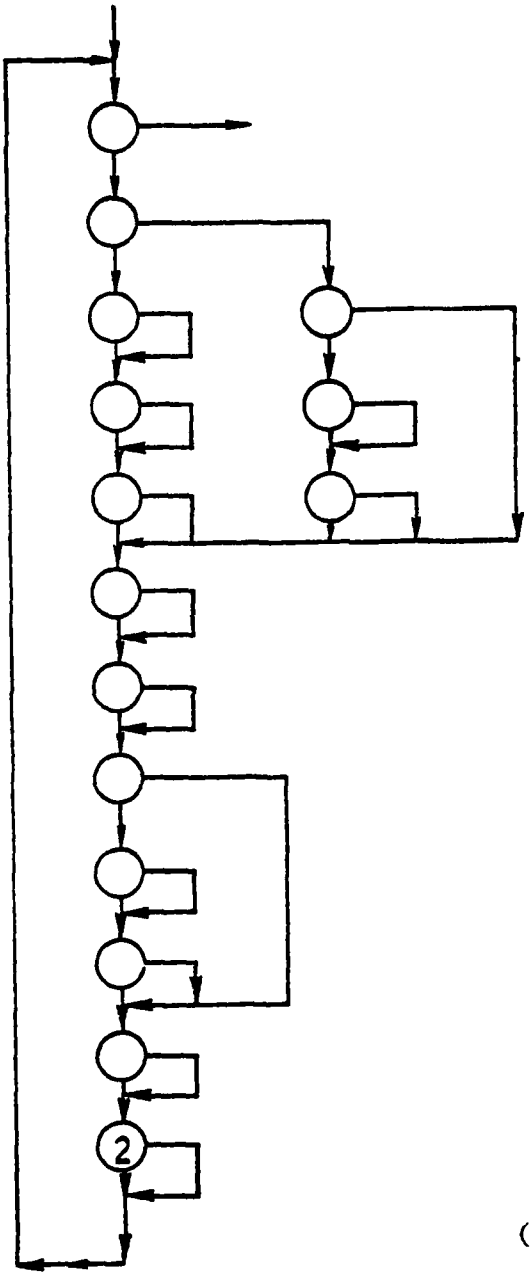
(d) P13_2A (e) P13_C

FIG. 36 FLOWCHARTS FOR THE SEVEN PROGRAMS



(f) P9_5

FIG. 36 FLOWCHARTS FOR THE SEVEN PROGRAMS (cont'd.)



(g) P18_6

FIG. 36 FLOWCHARTS FOR THE SEVEN PROGRAMS (cont'd.)

TABLE 1

Program	Completion Time (min)	# of Statements	Minutes/ Statements	Cyclomatic	Extended	MIN	Polynomial	# of Paths
P13_1B	60	15	4	4	4:6	4	$x+2$	4
P13_1A	70	12	5.8	5	5:24	5	$x+3$	5
P12_7	210	34	6.2	8	8:14	7	$3x^2+x$	14
P13_2A	240	33	7.3	9	9:11	-	$3x^3+7x^2+5x+1$	63
P13_1C	240	23	10.4	6	6:6	-	x^5+x^4	48
P9_5	630	54	11.7	13	13:7	-	$x^4+4x^3+7x^2+6x+2$	90
P18_6	860	53	16.2	16	16:17	-	$x^{10}+2x^9+2x^8+3x^7+2x^6+x^5+x^4$	3124

14.0 REFERENCES

1. L.A. Belady, "On Software Complexity," Proceedings of the Workshop on Quantitative Software Models, IEEE, pp. 90-94; 1979.

2. T.J. McCabe, "A Complexity Measure," IEEE Trans. on Software Eng., vol. SE-2, no. 4, pp. 308-320; December 1976.

3. J.C. Zolnowski and D.B. Simmons, "Measuring Program Complexity," Proceedings of the 1977 Fall COMPCON, IEEE, pp. 366-340; 1977.

4. M.H. Halstead, "Elements of Software Science," Chapters 3 and 8, Elsevier North-Holland, Inc., New York, NY; 1977.

5. M.L. Shooman and A. Laemmel, "Statistical Theory of Computer Programs-Information Content and Complexity," Proceedings of the 1977 Fall COMPCON, IEEE, pp. 341-347; 1977.

6. B. Curtis, S.B. Sheppard, P. Milliman, M.A. Borst, and T. Love, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Trans. on Software Eng., vol SE-5, no. 2, pp. 95-104; March 1979.

7. B. Curtis, "In Search of Software Complexity," Proceedings of the Workshop on Quantitative Software Models, IEEE, pp. 95-106; 1979.

8. G.J. Myers, "An Extension to the Cyclomatic Measure of Program Complexity," SIGPLAN Notices, pp. 61-64; October 1977.

9. V.R. Basili and R.W. Reiter, Jr., "Evaluating Automatable Measures of Software Development," Proceedings of the Workshop on Quantitative Software Models, IEEE, pp. 107-116; 1979.

10. E.T. Chen, "Program Complexity and Programmer Productivity," IEEE Trans. on Software Eng., vol SE-4, no. 3, pp. 187-194; May 1978.

11. M.R. Woodward, M.A. Hennel, and D. Hedley, "A Measure of Control Flow Complexity in Program Text," IEEE Trans. on Software Eng., vol. SE-5, no. 1, pp. 45-49; January 1979.

12. A.L. Baker and S.H. Zweben, "A Comparison of Measures of Control Flow Complexity," IEEE Trans. on Software Eng., vol. SE-6, no. 6, pp. 506-512; Nov. 1980.

13. M.L. Shooman, "Software Engineering: Reliability, Design Management," McGraw Hill Book Co., New York, NY, Fig. 4-15; 1981.

14. M. Lipow, "Application of Algebraic Methods to Computer Program Analysis," Report TRW-55-73-10, TRW Software Series; May 1973.

15. H. Ruston, "Programming with PL/I," McGraw Hill Book Company, Inc., New York, NY; 1978.

APPENDIX

We will consider two different flowcharts, characterized by the same polynomial and investigate their complexities. We will use to this end the flowcharts of Figures 31 and 32, redrawn with more detail in Figures A.1 and A.2.

The program segment describing Fig. A.1 is

```
IF A THEN IF B THEN IF C THEN IF D THEN IF E THEN S1;
                                                ELSE S2;
                                        ELSE S3;
                                ELSE S4;
                        ELSE S5;
                ELSE IF F THEN IF G THEN S6;
                                ELSE S7;
                        ELSE DO;
                                IF H THEN S8;
                                     ELSE S9;
                                IF I  THEN S10;
                                     ELSE S11;
                            END;
```

The flowchart of Fig. A.2 is described by

```
IF A THEN S1;
     ELSE IF B THEN S2;
                ELSE IF C THEN S3;
                          ELSE DO;
                                   IF D THEN S4;
                                        ELSE S5;
                                   IF E THEN
                          IF F THEN IF G THEN S6;
                                     ELSE S7;
                               ELSE S8;
                                          ELSE S9;
                                END;
```

To compare the complexities of these two program segments we will consider the conditions and the resulting executions.

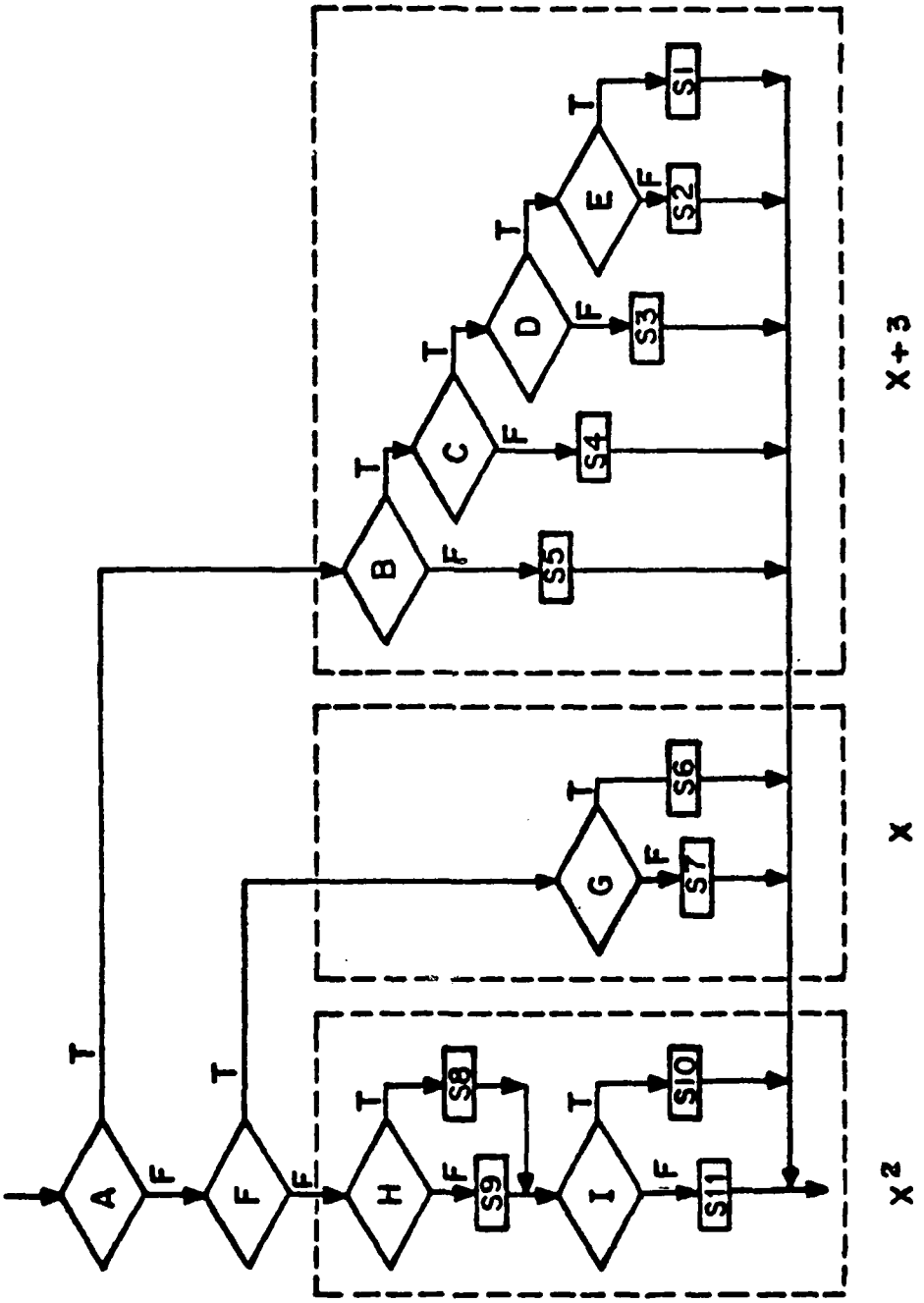


FIG. A.1 THE FLOWCHART OF FIG. 31 WITH MORE DETAILS SHOWN

APPENDIX

We will consider two different flowcharts, characterized by the same polynomial and investigate their complexities. We will use to this end the flowcharts of Figures 31 and 32, redrawn with more detail in Figures A.1 and A.2.

The program segment describing Fig. A.1 is

```
IF A THEN IF B THEN IF C THEN IF D THEN IF E THEN S1;
                                                ELSE S2;
                                        ELSE S3;
                                ELSE S4;
                        ELSE S5;
                ELSE IF F THEN IF G THEN S6;
                                ELSE S7;
                        ELSE DO;
                                IF H THEN S8;
                                     ELSE S9;
                                IF I  THEN S10;
                                     ELSE S11;
                            END;
```

The flowchart of Fig. A.2 is described by

```
IF A THEN S1;
     ELSE IF B THEN S2;
                ELSE IF C THEN S3;
                           ELSE DO;
                                    IF D THEN S4;
                                         ELSE S5;
                                    IF E THEN
                           IF F THEN IF G THEN S6;
                                      ELSE S7;
                                ELSE S8;
                                           ELSE S9;
                                END;
```

To compare the complexities of these two program segments we will consider the conditions and the resulting executions.

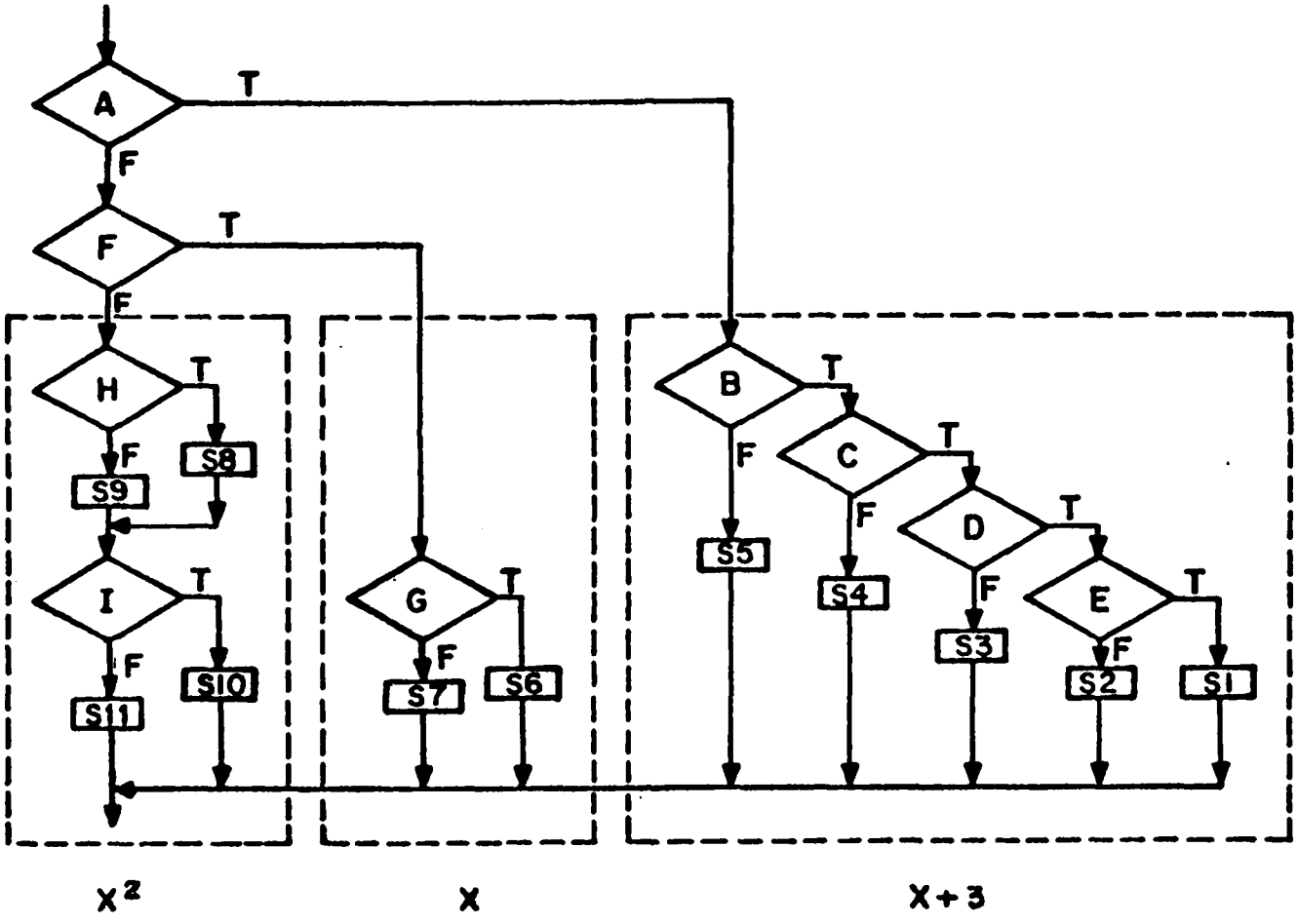


FIG. A.1 THE FLOWCHART OF FIG. 31 WITH MORE DETAILS SHOWN

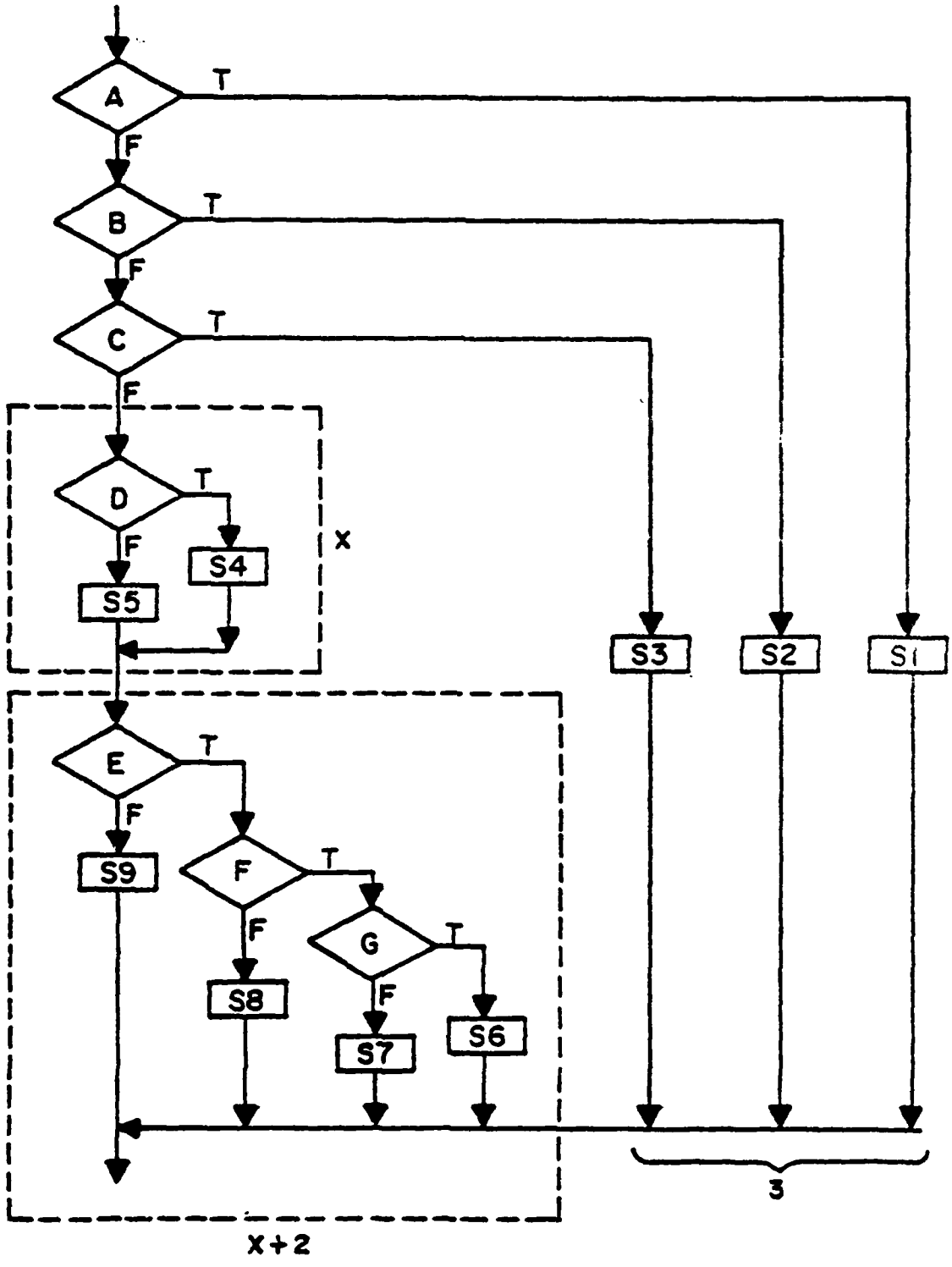


FIG. A.2 THE FLOWCHART OF FIG. 32 WITH MORE DETAILS SHOWN

For Fig. A.1, we obtain the listing

Condition	Execution of
ABCDE	S1
ABCD$\bar{E}$	S2
ABC$\bar{D}$	S3
AB$\bar{C}$	S4
A$\bar{B}$	S5
$\bar{A}$FG	S6
$\bar{A}$F$\bar{G}$	S7
$\bar{A}\bar{F}$HI	S8 & S10
$\bar{A}\bar{F}$H$\bar{I}$	S8 & S11
$\bar{A}\bar{F}\bar{H}$I	S9 & S10
$\bar{A}\bar{F}\bar{H}\bar{I}$	S9 & S11

For Fig. A.2 the listing is

Condition	Execution of
A	S1
$\bar{A}$B	S2
$\bar{A}\bar{B}$C	S3
$\bar{A}\bar{B}\bar{C}$DEFG	S4 and S6
$\bar{A}\bar{B}\bar{C}$DEF$\bar{G}$	S4 and S7
$\bar{A}\bar{B}\bar{C}$DE$\bar{F}$	S4 and S8
$\bar{A}\bar{B}\bar{C}$D$\bar{E}$	S4 and S9
$\bar{A}\bar{B}\bar{C}\bar{D}$EFG	S5 and S6
$\bar{A}\bar{B}\bar{C}\bar{D}EF\bar{G}$	S5 and S7
$\bar{A}\bar{B}\bar{C}\bar{D}E\bar{F}$	S5 and S8
$\bar{A}\bar{B}\bar{C}\bar{D}\bar{E}$	S5 and S9

which leads to the following observations:

1. Both flowcharts result in the same number of distinct actions (this being 11 for both flowcharts).

2. The flowchart of Fig. A.1 is realized with 13 statements, the flowchart of Fig. A.2 with 11 statements.

3. The total number of individual conditions to be satisfied for the execution of the 11 distinct actions in Fig. A.1 is 41 or 41/11 = 3.73 average conditions/actions. To execute the 11 actions in Fig. A.2 we must satisfy a total of 56 individual conditions, or 56/11 = 5.09 average conditions/actions.

This leads to the conclusion that from the first observation both flowcharts have the same complexity. The second observation tells that the flowchart of Fig. A.1 requires a program segment 18% [i.e., (13-11)*100/11] longer than the flowchart of Fig. A.2. The third observation tells that the testing of all actions will require 1.36 (i.e., 5.09-3.73) more conditions per action. Thus, Fig. A.1 is more complex than Fig. A.2 from one viewpoint and less complex from another viewpoint, but equally complex from the important comparison of distinct actions. The overall conclusion is then that both flowcharts exhibit the same complexity.